Calhoun: The NPS Institutional Archive

DSpace Repository

Theses and Dissertations | 1. Thesis and Dissertation Collection, all items

1998-03-01

The design and implementation of the Petite Amateur Naval Satellite (PANSAT) user services software

Hunter, George Kenneth

Monterey, California. Naval Postgraduate School

http://hdl.handle.net/10945/8222

This publication is a work of the U.S. Government as defined in Title 17, United States Code, Section 101. Copyright protection is not available for this work in the United States.

Downloaded from NPS Archive: Calhoun

http://www.nps.edu/library

Calhoun is the Naval Postgraduate School's public access digital repository for research materials and institutional publications created by the NPS community. Calhoun is named for Professor of Mathematics Guy K. Calhoun, NPS's first appointed -- and published -- scholarly author.

Dudley Knox Library / Naval Postgraduate School
411 Dyer Road / 1 University Circle
Monterey, California USA 93943

NPS ARCHIVE
1998.03
HUNTER, G.

DUDLEY KNOX LIBRARY
NAVAL POSTGRADUATE SCHOOL
MONTEREY CA 93943-5101

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

THE DESIGN AND IMPLEMENTATION OF THE PETITE AMATEUR NAVAL SATELLITE (PANSAT) USER SERVICES SOFTWARE

by

George Kenneth Hunter

March 1998

Thesis Advisor: Man-Tak Shing

Approved for public release; distribution is unlimited.

DUDLEY KNOX LIBRARY
NAVAL POSTGRADUATE SCHOOL
MONTEREY CA 93943-5101

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.

1. AGENCY USE ONLY *(leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	March 1998	Master's Thesis

4. TITLE AND SUBTITLE
THE DESIGN AND IMPLEMENTATION OF THE PETITE AMATEUR NAVAL SATELLITE (PANSAT) USER SERVICES SOFTWARE

5. FUNDING NUMBERS

6. AUTHOR(S)
George Kenneth Hunter

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT ***(maximum 200 words)***

PANSAT is an experimental spread spectrum, store-and-forward communications micro satellite. The Chief of Naval Operations C4I staff (N6) sponsors the project in order to determine the feasibility and effectiveness of using such a low-cost satellite to augment or eventually replace the existing military satellite communications architecture. While more than eight years of work has gone into the project, most of the nearly sixty theses thus far have dealt with hardware development. Prior to this thesis, the operations of the satellite were not formally defined, nor the desired software experiments specified.

This thesis develops a detailed definition of the communications software and operating parameters for PANSAT. The formally specified communications software provides electronic mail, binary file transfer, and direct real-time information exchange. This research also designs and develops experimental features which are non-existent on current micro satellites. The new features included provide the spacecraft with a pseudo positional awareness for a system with no sensor support for such, implement a new application layer protocol to optimize data communications, and perform self analysis to find and correct the effects of space anomalies in conjunction with a ground station.

This thesis also implements a subset of the formally specified software for initial operations to begin with spacecraft's launch in October of 1998. Further implementation and refinement will be based on actual operational results from PANSAT.

14. SUBJECT TERMS
PANSAT, User Services, Spacecraft Engineering, Amateur Satellite Communications, Amateur Radio Service, Ground Station, Software Engineering, NPSterm, Fault Tolerance

15. NUMBER OF PAGES
297

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

NSN 7540-01-280-5500

Standard From 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18
298-102

DUDLEY KNOX LIBRARY
NAVAL POSTGRADUATE SCHOOL
MONTEREY, CA 93943-5101

Approved for public release; distribution is unlimited

THE DESIGN AND IMPLEMENTATION OF THE PETITE AMATEUR NAVAL SATELLITE (PANSAT) USER SERVICES SOFTWARE

George Kenneth Hunter
Lieutenant, United States Navy
B.S., United States Naval Academy, 1990

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
March 1998

998.03
HUNTER, G.

DUDLEY KNOX LIBRARY
NAVAL POSTGRADUATE SCHOOL
MONTEREY CA 93943-5101

ABSTRACT

PANSAT is an experimental spread spectrum, store-and-forward communications micro satellite. The Chief of Naval Operations C^4I staff (N6) sponsors the project in order to determine the feasibility and effectiveness of using such a low-cost satellite to augment or eventually replace the existing military satellite communications architecture. While more than eight years of work has gone into the project, most of the nearly sixty theses thus far have dealt with hardware development. Prior to this thesis, the operations of the satellite were not formally defined, nor the desired software experiments specified.

This thesis develops a detailed definition of the communications software and operating parameters for PANSAT. The formally specified communications software provides electronic mail, binary file transfer, and direct real-time information exchange. This research also designs and develops experimental features which are non-existent on current micro satellites. The new features included provide the spacecraft with a pseudo positional awareness for a system with no sensor support for such, implement a new application layer protocol to optimize data communications, and perform self analysis to find and correct the effects of space anomalies in conjunction with a ground station.

This thesis also implements a subset of the formally specified software for initial operations to begin with spacecraft's launch in October of 1998. Further implementation and refinement will be based on actual operational results from PANSAT.

TABLE OF CONTENTS

I. INTRODUCTION 1
A. THE PANSAT PROJECT OVERVIEW 1
B. PANSAT USER SERVICES 4

II. BACKGROUND ON PREVIOUS WORK 5

III. DESCRIPTION OF USER SERVICES SOFTWARE 7
A. INTRODUCTION 7
B. SOFTWARE OVERVIEW 7
C. SPACECRAFT MODULE DESCRIPTION 8
D. GROUND STATION MODULE DESCRIPTION 13

IV. SOFTWARE REQUIREMENTS SPECIFICATION 19
A. INTRODUCTION 19
1. Purpose 19
2. Scope 19
3. Overview 20
B. GENERAL DESCRIPTION 20
1. Product Perspective 20
a. Software Description 20
b. End Users 22
2. Product Functions 23
a. Spacecraft Module Functions 23
b. Ground Station Module Functions 28
3. User Characteristics 31
4. General Constraints 31
a. Timing Constraints 31
b. Spacecraft Module Constraints 31
c. Ground Station Module Constraints 32
5. Assumptions 32
C. SPECIFIC REQUIREMENTS 32
1. Spacecraft Module 33
a. BBS Functions 33
b. Housekeeping Functions 74
c. Special Notes 80
2. Ground Station Module 81
a. Display Terminal Functions 81
b. Control Terminal Functions 96
c. Linux Terminal Functions 136
d. Server Functions 138

V. SOFTWARE DESIGN 155
A. USE CASES 155
1. Spacecraft Module Use Cases 156
2. Spacecraft Module Use Case Diagram 181
B. SYSTEM CONTEXT MODELS 187
1. Spacecraft System Context Model 187
2. Ground Station System Context Model 188
C. SUBSYSTEM MODELS 189
1. Spacecraft Subsystem Model 189
2. Ground Station Subsystem Model 190

VI. ENHANCEMENTS TO PANSAT MICRO SATELLITE SYSTEM 191
A. POSITIONAL AWARENESS 191
1. Introduction 191
2. Background of Orbital Mechanics 193
3. Implementing the Solution 196
B. FAULT TOLERANCE PLAN 198
1. Introduction 198
2. System Evaluation 200
3. Background of Previous Work 202
4. Error Classification 203
5. System Errors 205
6. Program Errors 209
7. Data Errors 211
8. Acceptance Tests 215
9. Conclusions 219
C. OPTIMIZED PROTOCOL 219
1. NPSterm Introduction 219
2. Bytecode Commands 221
3. Compression Algorithm 223

VII. IMPLEMENTATION ISSUES 225
A. REAL TIME TESTING 225
B. TEST PLAN 227
1. Satellite Module 228
2. Ground Station Module 237
C. PROGRAM SETUP 241
1. Satellite Module 241
2. Ground Station Module 242

VIII. CONCLUSIONS AND RECOMMENDATIONS 245
A. FURTHER WORK REQUIRED 245
B. LESSONS LEARNED 246
C. CONCLUSION 247

APPENDIX A. SELECTED SOURCE CODE EXTRACTS 249
A. POSITION DETERMINATION CODE 249
B. COMPRESSION CODE 257

APPENDIX B. SOURCE CODE ORGANIZATION 271

LIST OF REFERENCES 275

INITIAL DISTRIBUTION LIST 277

LIST OF FIGURES

Figure 1 - PANSAT representation 2
Figure 2 - Spread Spectrum Power Distribution 3
Figure 3 - The Ground Station Network Logical Organization 22
Figure 4 - Spacecraft Module Use Cases 181
Figure 5 - Spacecraft User Services Context Diagram 187
Figure 6 - Ground Station User Services Context Diagram 188
Figure 7 - Spacecraft Module Subsystem Diagram 189
Figure 8 - Ground Station Subsystem Diagram 190
Figure 9 - Orbital Coordinate System and Keplerian Elements 194
Figure 10 - Real Time Data Processing Testing Results 226
Figure 11 - Ground Station Initialization Directory Structure 243

LIST OF TABLES

Table 1 - General user BBS commands. . . . . -24-

Table 2 - Additional BBS commands for the Ground station only. . . . . -27-

Table 3 - User services control settings, with default values. . . . . -62-

Table 4 - Archive file extensions. . . . . -141-

Table 5 - Software Fault Tolerant Error Classes . . . . . -204-

Table 6 - Fault Tolerant Acceptance Test Classifications . . . . . -215-

Table 7 - NPSterm Command Bytecodes . . . . . -222-

LIST OF ABBREVIATIONS, ACRONYMS, AND SYMBOLS

The following list consists of abbreviations, acronyms and/or symbols that are used throughout this thesis.

ASCII	American Standard Code for Information Interchange - standard text representation on computers
AX.25	Link-layer packet-switching radio communications protocol used by PANSAT
BBS	Bulletin Board System
BIOS	Basic Input/Output System - ROM embedded software
BPS	Bits per Second - data transfer rate
CPU	Central Processing Unit - the "brain" of a computer
CS	Computer Science
CTRL	Control key - is pressed simultaneously with the next key listed
FAT	File Allocation Table
FCC	Federal Communications Commission
FTP	File Transfer Protocol
GUI	Graphical User Interface
HAM	Amateur radio, governed by the FCC, derived from a term meaning poor operator
HTML	Hyper Text Mark-up Language, text format used for WWW pages.
kB	Kilobytes (or 1,024 bytes)
LAN	Local Area Network
MB	Megabytes (or 1,048,576 bytes)
Mbps	Mega bits per second, a data transmission rate
MHz	MegaHertz or frequency in millions of cycles per second
MFC	Microsoft Foundation Classes - Windows programming library
NASA	National Aeronautics and Space Administration

NPS	Naval Postgraduate School, Monterey California
NPSterm	One of two possible Application-layer communication protocols used by PANSAT (the other is ASCII character streams)
PANSAT	Petite Amateur Navy Satellite
RAM	Random Access Memory
ROM	Read Only Memory
SCOS	Spacecraft Operating System, written by the BekTek corporation
SCC	Serial Communications Controller
SSAG	Space Systems Academic Group
SRS	Software Requirements Specification
TNC	Terminal Node Controller
UHF	Ultra High Frequency
WWW	World Wide Web

ACKNOWLEDGMENTS

I would like to thank Glenn Harrell for recruiting me into this project. Even though the chain of events leading up to me meeting Glenn seems nearly impossible, Glenn says "there are no accidents" and I believe him.

My sincerest appreciation goes to the guidance of my thesis advisor, Man-Tak Shing. My tendency is to try to do everything. Man-Tak put blinders on me and forced me to focus on a tangible amount of work, of which I could produce a viable product. Without Man-Tak, I don't know if I would ever finish working on this project.

Although labeled as a second reader, Jim Horning was much, much more. All of the work done in this thesis was done in collaboration with Jim. As the center of focus for all software engineering for PANSAT, Jim provided me ideas, guidance, assistance, coaching, honest feedback, and patience, as well as devoting an tremendous amount of time to working on this thesis. None of my work would have been possible without Jim.

I owe a large gratitude to James Hetfield for, through his words, unknowingly providing the inspiration and motivation I needed to press forward through the dreary part of researching and compiling this thesis.

Despite all these people I may thank, none of this would be have been worth the effort without the loving support of my wife. Acting as my "sounding board"and her enormous amount of proofreading, Antonietta was essentially my third reader. Furthermore, she patiently endured the countless hours when I was home, but not actually there since I working on this project. I can never repay what she has given me.

I. INTRODUCTION

A. THE PANSAT PROJECT OVERVIEW

Since 1989, the Space Systems Academic Group (SSAG) at the Naval Postgraduate School (NPS) has been developing PANSAT, a small communications satellite. The acronym PANSAT stands for Petite Amateur Navy Satellite.

Under the auspices and sponsorship of the Navy Space Systems Division (N63), the PANSAT project fulfills a threefold objective. First, and primarily, the satellite is a proof-of-concept for a low-cost, packetized, spread spectrum communications system. By implementing and testing PANSAT, the capability to enhance military communications using a small satellite will be evaluated.

The means to evaluate the first objective is actually encapsulated within the second objective. PANSAT will provide store-and-forward communications for the amateur radio community, commonly known as HAM radio. HAM operators have been involved with packet radio and satellite communications since the late 1960's. By providing this community with the PANSAT facility, HAM operators will be increasing their resources of a familiar functionality - store and forward satellite communications. Moreover, they will be exploring and testing a new mechanism to achieve this functionality (*spread spectrum* communications). Thus, while the HAM community receives a free new resource to experiment with, the SSAG obtains a vast knowledgeable user base to test and assess the platform. Furthermore, this PANSAT evaluation will be conducted without impacting current military communication facilities or operations, because it will use UHF amateur radio frequencies.

The third objective of PANSAT is to enhance the education of military officers at NPS through the development of a digital communications satellite. Since the project's conception, nearly sixty PANSAT related theses have been completed. Furthermore, in this day and age, satellites are a mainstay for military communications and operations. Once the satellite is in orbit, it can be used as a "space-based instructional laboratory" for military officers to learn more about the mechanics of satellite communications. The

concepts learned while experimenting with PANSAT should prove extremely valuable in understanding current communication operations.

Although labeled as "petite," the actual NASA classification for the satellite is "micro." Once completed, the twenty-six sided spacecraft will weigh 150 pounds and be nineteen inches in diameter. It will be a tumbling satellite with no attitude control or means of propulsion. PANSAT will continually work off of a pair of battery packs. When the Earth is not eclipsing the sun from the spacecraft, solar panels will recharge those batteries. To accommodate the tumbling nature of the satellite, solar panels will encompass almost the entire spacecraft's outer shell, providing power no matter what the spacecraft's attitude is. Also, four omni-directional antennas will allow communications with Earth no matter what the orientation is (see Figure 1 for spacecraft representation).

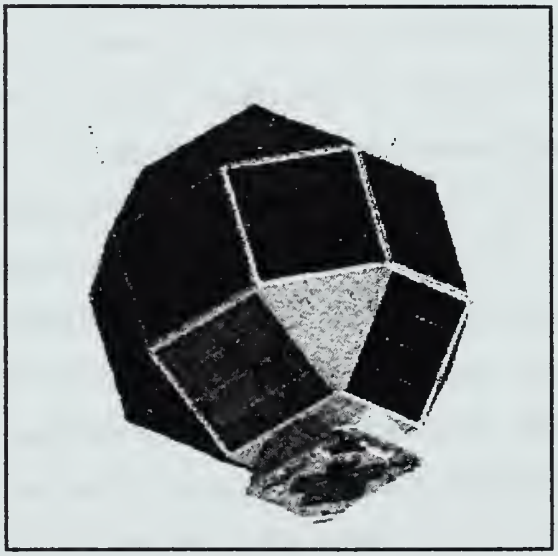


Figure 1 - PANSAT representation

PANSAT will be the first satellite of its class to utilize spread spectrum communications. Direct sequence spread spectrum modulation is a technique that spreads a conventional narrowband signal by mixing it with a bit stream over a wider frequency band in which the energy at any given frequency is much lower (see Figure 2). The result is a dilution of the signal energy with respect to bandwidth. The spread spectrum signal

has the same energy per bit as the narrowband signal, but the power density at any one frequency is significantly lower. The signal can be spread to such a point that it is entirely below the noise level of a conventional receiver. Spread spectrum modulation provides the advantages of low probability-of-intercept, low probability-of-detection, resistance to jamming and low probability-of-interference (to and from other users in the band).

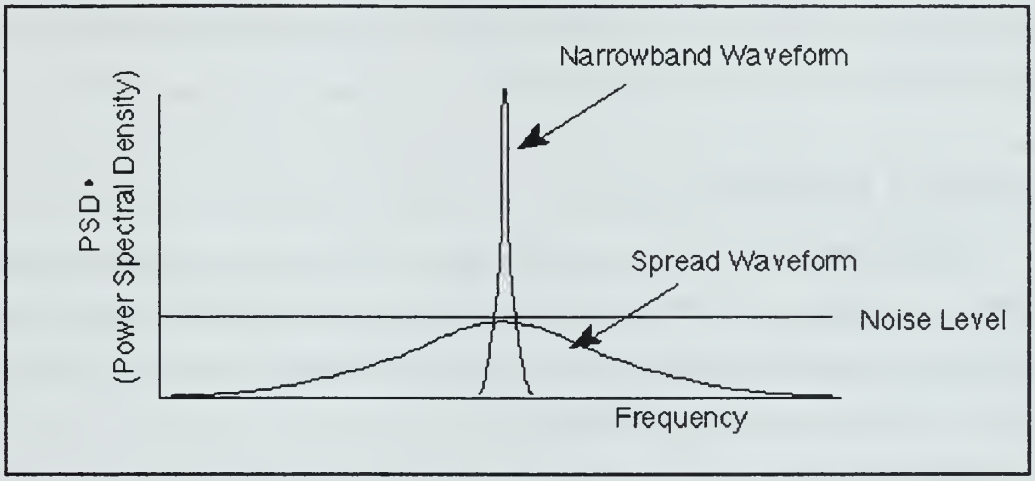


Figure 2 - Spread Spectrum Power Distribution

PANSAT will communicate at a center frequency of 436.5 MHz and at a data rate of 9.84 kilobits per second. The data link layer protocol used for communication will be the amateur packet radio protocol AX.25, the most widely used means for amateur packet radio communications.

Since the satellite is orbiting the Earth, the window of opportunity to communicate with PANSAT is limited. The particulars every time the spacecraft is within line-of-sight of a ground station will be different (i.e. time satellite is line-of-sight and position on the horizon). A window of opportunity can last from approximately two to ten minutes, averaging around six minutes. Additionally, the windows will only occur a few times a day. Once the spacecraft is launched from the Space Shuttle, PANSAT will be tracked and an actual orbit will be established. With this established, the actual windows of opportunity can be pre-determined.

PANSAT is scheduled to be launched from the Space Shuttle in October of 1998 as a secondary payload through the NASA Hitchhiker Program.

The Hitchhiker program was established to allow for low-cost and quick-reaction accommodation of secondary payloads on the Space Shuttle. The Hitchhiker carriers can carry payloads side mounted in the shuttle payload bay or mounted on a cross-bay "bridge" structure. PANSAT will be mounted within a canister to a small spring loaded ejection mechanism. When the shuttle is positioned at the appropriate orbital position pyrotechnic bolts will fire releasing the satellite.

B. PANSAT USER SERVICES

PANSAT will provide a store and forward communications capability to HAM operators. This functionality will be implemented in an uploaded software package, only loaded once the spacecraft is orbiting and operational. The services that this uploaded software provides is the scope of this thesis.

The satellite that is launched will have practically no functionality at the user level. Instead, programmed in its ROM, will be a routine such that as soon as the satellite becomes operational, it will try to establish contact with the ground station. Once contact is established, the ground station will upload the user services software, which the satellite will execute.

Once the user services software is loaded, it will allow any HAM operator in the world to upload and download mail messages, as well as binary files. Also, the operators will be able to check the satellite telemetry, both current and historical. The user will also be able to immediately communicate with any other user that is currently in communications with PANSAT. Essentially the satellite will serve as a sort of bulletin board system (BBS), albeit one with enhanced functionality.

The user services software not only implements this BBS system onboard the satellite, but also implements the ground station to remotely manage the satellite from NPS. A full description of the user services functionality is detailed in Chapter III of this thesis.

II. BACKGROUND ON PREVIOUS WORK

While numerous theses have been completed on the hardware aspect of PANSAT's design, construction and testing, very little has been written about the software side of the project. For the most part, the requirements of software have been ideas held by the members of the SSAG rather than being formally specified. Thus far, only two theses have held more than a paragraph description of the user services software.

A brief description of user services is provided by Fred Severson in his 1995 thesis "An Overview of the Petite Amateur Navy Satellite (PANSAT) Project." However most of the short section of user services discusses communications protocol and means rather than the idea of what will be communicated. This section did provide a preliminary understanding of the purpose and functionality of PANSAT. The complete role of the satellite was determined in extensive interviews with Jim Horning and Dan Sakoda of the SSAG and is detailed in Chapter III of this thesis.

Gregory Lawrence in his 1994 thesis "Preliminary PANSAT Ground Station Software Design and Use of an Expert System to Analyze Telemetry" does not focus on the user services portion of the ground station software. Rather, he concentrates on an artificial intelligence means of analyzing telemetry data retrieved from PANSAT. While the concepts presented in this thesis will be used in the telemetry display window on a ground station terminal, the program itself cannot be incorporated into the ground station. Lawrence's program was written in the Prolog computer language, which cannot be integrated with the C++ program the ground station will be written in.

Beyond the scope of work that has been explicitly done for PANSAT, the commercial realm has obviously been working on various sorts of satellite projects for years. In fact, several programs already exist which provide *at least some* of the rudimentary elements that will be incorporated in PANSAT's user services functionality. Unfortunately, the companies that created these programs will not release their source code or methods in order for them to retain software proprietary. Thus the satellite's software building blocks must be designed and created from scratch.

The result of this minimal available background of work is that the design and implementation detailed in this thesis is not forthcoming from or reliant on any other body of work. This work, however, will form the foundation for future work on the PANSAT user services software as well as any experiments that may be conceived during the spacecraft's lifetime.

III. DESCRIPTION OF USER SERVICES SOFTWARE

A. INTRODUCTION

As denoted in Chapter II, there did not exist a complete concept of the user services software. Consequently, the following description was based on a series of meetings between myself and Jim Horning from the SSAG. The initial meetings took place in January of 1997, but the document continued to mold and take shape for the remainder of the 1997 calendar year.

This narrative was created to sum up the features and functionality of the PANSAT User Services Software package. After the contents were agreed upon by the members of the SSAG, this was used as the basis for Chapter IV, the formal requirements specification.

Although the narrative is in paragraph format, each paragraph is organized as a list of functionality rather than a pure narrative. Each paragraph merely describes a program feature or group of features. The narrative starts out overviewing the entire system, then the spacecraft module and ground station module functions are separately described.

B. SOFTWARE OVERVIEW

The user services software is required to implement the ground station and onboard satellite bulletin board system operations, as well as performing satellite "housekeeping" functions detailed later. Existing onboard software, some of which will be interfaced by user services, already provides other housekeeping operations, as well as the operating system, telemetry features, file system, packet control, battery operations, automatic log keeping, etc.

Throughout this narrative, the term "ground station" will refer only to the HAM radio site operated by the Space Systems Academic Group at NPS. This site is the only operational control node for the satellite. All other end users which may access the satellite will be referred to as "user." These "users" could be any amateur HAM radio operators located throughout the world. "Users" can utilize the services provided by the

satellite, but cannot change PANSAT's control settings. Only the ground station can modify the control settings. The difference in the options provided to these two different types of end users is provided throughout this narrative.

The software required for the user services is divided into two separate modules. One part will be the ground station. A public domain user interface program will be extracted from elements of the ground station module and be made available via the Internet for downloading. This general user module will just have typical BBS user interface features - all the ground station controlling elements will be removed.

The second part of user services software will be the software onboard the satellite which provides BBS functionality. Once initial contact is achieved between the satellite and the ground station, the ground station will upload the software for user services. PANSAT will then execute this program, which will provide the user interface and functionality with the satellite.

Additional features to both modules will be added as the project matures and as new requirements develop after interacting with the functional satellite.

C. SPACECRAFT MODULE DESCRIPTION

The first module described is the system onboard the satellite. This program needs to run on Intel's 80C186 processor, executing on top of BekTek's spacecraft operating system (SCOS) and utilizing BekTek's AX.25 protocol utilities. Interfaces to both of these units are via Microsoft C version 5.0/5.1 object files. The user services program will most likely have about 300K out of 512K of memory left over for combined code and data, but the goal is to have the program use as little memory as possible.

PANSAT will normally be in a receive-mode, waiting for a request-to-connect command from a user. Even when a connection is ongoing, the satellite will concurrently wait for another user to connect, until the maximum number of connections has been made. A user can connect with, or log onto, the spacecraft in either one of two modes. The first mode is via a generic ASCII interface, the standard BBS method for HAM operators. In this mode, all interaction between PANSAT and the user are via ASCII data

streams in a purely character environment. The second type of connection shall be called NPSterm, an NPS-propriety application level protocol. This is a GUI based interface to PANSAT and available only via the ground station software, or the user package derived there from. This connection will use shortcuts and data compression in transmissions. This decreases the interaction between the user and satellite - the less data transferred, the more utilization of the limited communications window is achievable. The user also works via a GUI interface to make things more robust and easier to use. The actual technical definition of NPSterm, which will still use/ride on top of the AX.25 data communication protocol, is defined in Chapter VI.

The major requirement for the spacecraft services module is to perform BBS-like features. It will allow up to 16 simultaneous connections, one of which is reserved for the ground station, implemented with interleaved concurrency. A connection is an established communications link between a user or ground station and satellite. The connect process is implemented similar to logging onto a computer terminal. "Logging onto" user services will be identified by the caller trying to connect to call sign "PANSAT". If the caller is using callsign "KD6CXV" (the default value), the caller will be identified as the ground station. The ground station callsign can be changed from the default value, but only by the ground station. The satellite will then provide the time of its most recent connection with the ground station. For both users and the ground station, PANSAT will print out a happy face symbol with a welcome to PANSAT greeting. The happy face will be constructed with a few telemetry values. Next, the system will check to see if the connection is by a callsign which was previously disconnected during a file transfer. If it is, PANSAT will prompt the user whether or not to continue with the file transfer, exactly at the point the previous operation left off. After the file transfer, the connection process will continue with the next step. To facilitate continued file transfer, PANSAT will keep track of interrupted connections for a duration and maximum allowable number set by the ground station. The last step of the connection/login process will be to display broadcast messages, which can only be set by the ground station.

If the means are developed, PANSAT should check a new connection's power (signal strength). If the connection is broadcasting over a ground station specified power level, PANSAT should warn the user. If the user doesn't lower the output level being used, PANSAT will disconnect that user. The too strong disconnect function is a secondary requirement.

The user menu will provide the following services: It will allow a user to disconnect. Additionally, the connection will be automatically disconnected if no input is heard from the user in 2 minutes or a time duration set by the ground station. It will allow users to obtain two types of telemetry. By getting "current telemetry," the values at that moment in time will be displayed and broken out (put in a readable format with headings). The "stored telemetry" of the past few days, however, must be downloaded in file format (the proper file will be automatically selected from the spacecraft's storage), and the information will be in raw data and must be deciphered by the user. Further, the ground station can get a quick dump of all the current control settings. The user will be able to send mail to a single user or to all users of the system. They will be able to read any mail message on the system. They will be able to upload a file and tag it for a specific user or users, or for every user of PANSAT. They will be able to download any file. Additionally, to save the time required for uploading, a user may forward any message or file to another user.

Anybody can access any of the general mail or files stored on the system. The difference in mail for a single user or for all is how the listings of the mail and files are made. The user can select to see a listing of every mail message on the system, just the messages directed at that user, just the messages directed at that user that have been posted within the last twenty-four hours, or just the messages directed at that user plus the messages sent to all. The same operations for mail exist for files, except where mail must be in text format, files can be in any format and are uploaded rather than directly entered in. Also, both mail and files may have a size limit set or disabled, by the ground station, the default limit is for files is 256 kB, for mail is 4 kB. To get a better idea on file size

limits, an initial experiment with the satellite will be to see how large a file can be put onto the system in three continuous days of connections.

A user may delete a mail message or a file only if it was sent to them or they originated it. The ground station may delete any or all mail or files. Further, the system will autodelete files according to priorities set by the ground station. The priorities will be based on a combination of mail/file size, length of time on the system, and the amount of available free (unused) storage space. Also, mail/files sent to all users will have separate autodelete settings from those mail/files only to a single user. This includes an additional setting of the maximum number of mail/files to "all type" messages allowed on the system, which is set by the ground station.

A user will be able to list all the other users currently connected to the system. That user will be able to send a one-line message to any other user currently connected, which will immediately be sent to the other user's terminal. Additionally, the sender can indicate the one-liner is for all, which will send the message to every user currently connected to PANSAT. For the machine, this is a high priority function, meaning it overrides other functions, but does not interrupt the other functions.

A help feature on all these functions will be online. It will be very minimal for the ASCII terminal, but much more extensive for the NPSterm because it will actually reside in the user's software. It will provide simple assistance to the BBS interface and functions.

The ground station may terminate the user services program. The program is normally in a perpetual loop, always providing its services and never terminating. However, this command would allow the program to exit gracefully, which involves closing out all current work and conducting "garbage collection" (resource reclaiming). By exiting gracefully, the user services will be able start up retaining all the mail/files from the previous session, resulting in a minimal loss of continuity. Once the program is terminated, all the memory used by the program will be released back to the operating system for reuse. To restart the process, the user services program needs to be re-uploaded by the ground station. This procedure would allow for an updated version of

the program to be loaded on PANSAT, then allow operations to continue where they left off.

Whenever a command reserved for the ground station is received, PANSAT will first check to ensure the command was received from the ground station connection. Next, the satellite will send a verification function back to the ground station. If the ground station responds with the correct answer, PANSAT will execute the command. For each and every ground station only command, a new verification set of numbers will be sent. If the ground station enters a general user command, however, no verification function will be queried.

Every time the ground station connects, the user services software will generate an entry, with a data/time stamp, to the log manager. Additionally, every time a failed ground station only command is attempted, this information with a date/time stamp is sent to the log manager. The log manager is a separate driver running about the spacecraft, the interface to which will be provided by the SSAG.

The second function of the user services will be to provide certain "housekeeping" operations. One such function will be a continuous calculation of the spacecraft's position. The ground station will periodically send an update of the position of the satellite to the spacecraft. Based on this, PANSAT will deadreckon its current position. When it determines that it is above water or areas that communication with the satellite is highly unlikely, the spacecraft will temporarily shut down non-essential equipment to conserve power, such as putting the modem in a listening duty-cycling mode. This function can be disabled by the ground station. Also, if no contact with the ground station has occurred in 24 hours, this function will be disabled. This deadreckoning feature is an experiment and a secondary requirement.

Another "housekeeping" function will be to check the storage system for file fragmentation. It will conduct this check during periods of no-likely communications. If it is above a ground station set threshold, the system will defragment the storage memory. This is a secondary requirement.

Finally, to conserve on storage space, the system may use a data compression scheme to save mail and files. This would be completely transparent to the users, but could be disabled by the ground station.

All of these "housekeeping" functions should be implemented as concurrent tasks to the normal BBS functions.

All the ground station settings should be updated in a single information block, rather than having to go through a full set of commands for every operation. The only additional ground station's function not already mentioned would be the setting of PANSAT's onboard time/date stamp, which would be part of the single information block.

If possible, the satellite module should use fault tolerance construction to be able to better survive the anomalies associated with space. This means the program will continually check itself for errors. If it finds one, it should try to correct itself, or notify the ground station if a reload of the software is necessary.

D. GROUND STATION MODULE DESCRIPTION

The second part of the user services is the ground station software. The bulk of this software needs to run on the Windows NT operating system on a PC. However, the PC controlling the modem will be implemented on a PC running the Linux operating system. The two operating systems will communicate with each other via the socket datagram protocol.

The complete ground station software package will be used only at the SSAG at NPS, but a subset of functions will be bundled into a general user's package. This user's package will incorporate an interface to all the spacecraft functionality listed in Section C above, except for the functions listed as ground station only. Only the windows in this section specifically listed as such are the modules included in the general user package. This package would be available for downloading to anybody in the world via the Internet.

The ground station will actually be operating on four computers simultaneously. The computers will be connected by a LAN using the TCP/IP protocol. One of the computers will be operating Linux, while the other three will use Windows NT.

The Linux machine will simply function as a relay between the satellite and the other ground station computers. This computer will serve as a Terminal Node Controller (TNC) emulator, providing AX.25 services and controlling the Serial Communications Controller (SCC) chip and modem. It will relay any packets received from PANSAT to other three computers. Any of these other computers can also send a packet to the Linux box, which it will then send to the satellite. All these packets, to and from PANSAT, will be backed up on a local hard drive for twenty-four hours. This will allow communication reconstruction with the satellite if required.

An additional function of the Linux box will be to rotate the antenna as necessary, utilizing data from the satellite tracking window (on the display terminal), for optimal reception with the satellite. Drivers for these functions will be provided by the SSAG.

The first Windows NT computer will be a display terminal with a large monitor. It will have a satellite tracking window, which will display a Mercator projection of the world, with the PANSAT's track and current position plotted. Color shading will denote the section of the satellite's track when the ground station is in its footprint. Also, the time for the next opportunity for the ground station to communicate with PANSAT will be displayed. This tracking window will also be included in the general user package.

The display terminal will also have a window showing the latest telemetry and status of the satellite. This information will be defined later, but should be depicted graphically (i.e. represent battery power in line graphs) and use colors to denote thresholds (i.e. green, yellow, red) as much as possible. The telemetry window will also be included in the general user package.

Another window for the display terminal will be a read-only "mirror" of the ongoing communication to and from the satellite. The window can be set to copy interaction between PANSAT and a single station (such as the ground station, which is the default), between multiple stations, or to copy all the communication to and from PANSAT. This window will also be included in the general user package.

The final window on the display terminal will be the archive management window. This window will allow the viewing, deleting, printing, and searching of the archived files.

These files will contain PANSAT's telemetry data and the satellite's image. An image is a duplication of as much of PANSAT's storage as possible. For instance, an image may contain a copy of all the mail and file listings, as well as any mail or file that was obtained by the ground station. These images are acquired whenever the ground station interacts with the spacecraft. While the system will not explicitly load mail or files just for the image, any time the ground station does a download for any other purpose, that download will be included in the image. The system will, however, obtain a complete directory for the image automatically with each connection. The images will be stored automatically by the system, using a date stamp hierarchy for storage - a year directory contains the month subdirectories which contains the day subdirectories, which contain all the files from a single day. This window should also be available on the control computer.

The second Windows NT computer that makes up the ground station will be the control terminal. The terminal will normally be in general access mode. If the operator of the terminal enters the super user password, however, the terminal will be put in super user mode. Super user mode will last for ten minutes, or until explicitly terminated by the user. The difference in the modes is that in super user mode, the operator can use the commands listed in section C above as for the ground station only. If in general access mode, only the user commands can be chosen. If in general access mode and one of the ground station commands is chosen, the operator will be warned. If in super user mode, when the operator uses one of the restricted commands, it will send the command to the satellite, then automatically respond to the satellite's verification function. The verification function responder will absolutely not be included in the general user package. Note that batch jobs, listed later, cannot use super user mode. These super user restricted commands must be done via active human intervention.

The key window on the control terminal is the interaction window. This window is the typical BBS user interface, however this version has the NPSterm GUI. A difference between the user interface version and the ground station version, is that the ground station terminal window will offer more options. The additional ground station options are described in section C above. The operator will be able to create mail

messages for posting to PANSAT, even while not connected to the satellite. When connection with PANSAT is established, the mail messages will be sent to the spacecraft for posting. Likewise, the operator may read downloaded mail messages immediately, or store them for reading after the connection with PANSAT has been terminated. This storing is separate from the archiving function.

Another window on this display will be the "control panel." This window will show all the settings controlled by user services for the satellite. The ground station user needs merely to modify one of the settings in this window and during the next connection, the entire setting block may be sent to the satellite, updating its settings. The terminal must be in super user mode, however, to both modify the settings and to send the settings up to PANSAT.

Also, the control terminal will be able to be set up to automatically do commands via a batch job. This batch job will be pre-compiled to check for validity. An example batch job could be to automatically log in to the satellite, download current telemetry, then logoff. This will allow for unattended interaction with PANSAT (such as operations at night) or for the automated beginning of a connection session before the terminal operator resumes control of the session. The batch command will be able to do any of the non-super user commands

On both versions however (ground station and general user), if NPSterm on the satellite is a more recent version than the user program, the program will warn the user of the need to update the software, then shift into the pure ASCII terminal mode. This will allow the connection to continue, but will not be in the enhanced interface of NPSterm.

The third Windows NT computer will perform as the network server. Additionally, it will collect the archive files, both the telemetry and images, as mentioned above. The archive files will be stored on this computer.

The final function of the ground station software would be to have an Internet interface with the satellite/project. This will be implemented on the server as well. Via the Internet, anybody can access PANSAT via a virtual HAM user site, or a web site that allows a non-HAM operator to legally perform the functionality of one. The functions

provided by this site include the ability to post a message on the satellite's BBS, look at any of the archived files, view any messages or telemetry data from the storage images as desired, or request a message be downloaded for viewing during the next pass. An Internet viewer could see the desired message in the image directories, but the actual file may not be downloaded yet. After each pass, these Internet pages are recompiled with the latest telemetry and satellite image.

A special feature for the Internet would be mail forwarding. As a mail message is being saved into the archive, the server will check the first line of the message. If the first line reads "Internet email: ..." the server will forward the mail message to the Internet address specified after the colon on the first line. These Internet access features are secondary requirements.

This ground station program will need to interface with other, as yet undefined, control modules. It may just be required to launch the execution of these other programs, which would use their own windows. Once such module is the program that performs the initial software upload. When contact with PANSAT is first made, this module will upload the user services module to the satellite for execution. None of the user services will be operational without this first step. Also, if PANSAT resets, for any reason, this module will need to be called to reload the spacecraft user services software.

IV. SOFTWARE REQUIREMENTS SPECIFICATION

A. INTRODUCTION

1. Purpose

This Software Requirements Specification (SRS), prepared in the format as specified in IEEE Std. 830-1993, modified as required to adhere to the Thesis Preparation Manual guidelines, establishes the requirements for the PANSAT user services. This document is intended to form the basis for the design of the software product.

2. Scope

The PANSAT user services project will:

- Provide software to run a bulletin board system (BBS) onboard a satellite, allowing amateur HAM radio operators to read telemetry, send and read mail or files, and directly communicate with other operators.
- Provide software that will perform "housekeeping" experiments on the satellite, which are briefly described in section B.2.a of this chapter.
- Provide software to remotely manage the user services system and archive data received from the satellite.
- Provide a software package deliverable to anybody in the world which would enable interaction with PANSAT.
- Provide access to the satellite's user services via the Internet.
- Define and utilize a new application-layer communications protocol, designated NPSterm.

The consumer for this software project is the Space Systems Academic Group (SSAG). The points of contact in SSAG for this project are Mr. Daniel Sakoda and Mr.

James Horning. The producer of the PANSAT user services software and documentation is LT Ken Hunter.

3. Overview

The remaining sections of this document are organized in the following manner:

➳ Section B of this chapter gives an overall description of the organization, functionality, and restrictions of the user services software.

➳ Section C breaks down the software functions into individual elements, then provides the specific requirements for each element.

One additional note: when sections are referred to in this specification, the sections implicitly refer to a section of this chapter.

B. GENERAL DESCRIPTION

1. Product Perspective

This section covers the basic precepts on which the software package is based, for each software component and as a whole.

a. Software Description

The software required for the PANSAT user services is divided into two separate entities: the spacecraft and ground station modules.

The first module is the spacecraft software. After initial contact with PANSAT is established, the ground station will upload this program. Prior to this upload, the satellite will have no user services functionality. Likewise, if PANSAT resets, this program will need to be reloaded because there is no permanent storage system on the satellite. After loading, the program will execute. The satellite will then be able to perform the BBS and "housekeeping" functions described in section B.2.a.

The second part of the user services software is the ground station module. This program will be used to manage the PANSAT user services system, as described in section B.2.b. Additionally, the ground station will have the following characteristics:

TNC Emulation - The software will act as a high-level Terminal Node Controller (TNC) emulator. That is, TNC equipment will not be necessary for this system. The software will provide the functionality typically found in this equipment, such as packeting of data and control of hardware.

Ground Station Composition - The ground station will actually be operating on four computers simultaneously. The computers will be connected and working together via a LAN using a TCP/IP network. Three of the computers will run Windows NT, the other will run Linux. The first NT computer will be a display terminal, which will incorporate a very large monitor. The purpose of this terminal is to view information relating to PANSAT without tying up the resources needed to manage the satellite. The second NT computer will be the control terminal. This terminal will have the resources to control/manage the satellite. The third NT computer will be the network server as well as manager of the Internet site set up to interact with PANSAT. To support this multi-computer architecture, the ground station module will actually be composed of several separate programs, one for each terminal. The network description for each terminal is depicted in Figure 3. The specific functionality incorporated in each program is listed in section B.2.b.

Ground Station Derivative - The complete ground station software package will be used only at NPS, but a subset of functions will be bundled into a general user's package. The user's package will have all the options

listed for a general user, however, all the commands and windows listed as for "ground station only" will be omitted. This program will be available for downloading to anybody in the world via the Internet.

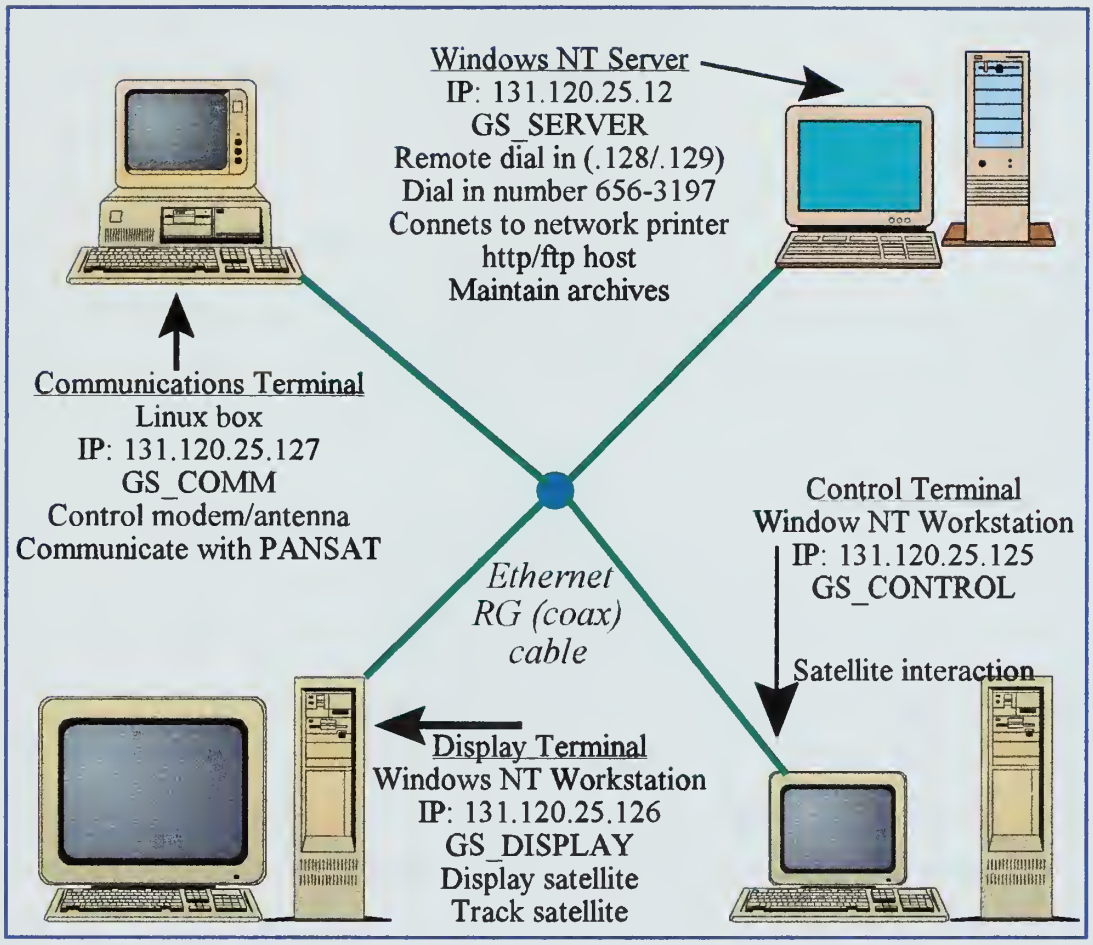


Figure 3 - The Ground Station Network Logical Organization (Domain name: GS)

b. End Users

There are two distinct types of end users of PANSAT's user services. A reference to "end user" in this document will refer to both types, otherwise the name of the specific type will be used. The first type, called the "ground station," is the single HAM radio site operated by the Space Systems Academic Group at NPS. This site is the one and only control node for the satellite. All other

operators which may access the satellite are designated "users." A "user" could be any amateur HAM radio operator in the world. "Users" are allowed to use the BBS services provided by the satellite, but will not be able to change PANSAT's control settings. Only the ground station can modify the control settings. The specific differences in the functions available to these two types of operators is provided in section B.2.b, with the difference in the commands available to each is described in Tables 1 and 2.

2. Product Functions

This section gives a brief description of all the functions for each component of the user services software. A more complete and detailed description for these functions is given in section 3 of this document.

a. Spacecraft Module Functions

There are two main parts to the spacecraft module, the bulletin board system (BBS) and the special experimental "housekeeping" functions.

The primary functionality of the PANSAT user services is the BBS. This is the interactive portion of the software, which is characteristically similar to a dial-in computer BBS. Up to 15 users will be allowed to be simultaneously connected to PANSAT, using an interleaved concurrency scheme. Further, an additional connection will be reserved for ground station use only, which will be identified by its callsign. After connecting (a description of which is in Table 1), PANSAT will send a greeting message to the connection containing a "happy face". This happy face will be composed of telemetry values, providing the HAM radio enthusiast with immediate status of the satellite. Next, the user or ground station will be allowed to continue a file transfer, if one had been interrupted during their previous connection. After the file transfer is complete, or immediately if the file transfer is not performed, PANSAT will send the connection a broadcast message. This message will be set by the ground station. Finally, the connection will be

offered a menu of choices, described in Table 1. After an option is performed, the menu is repeated until the disconnect command is received. In addition to the Table 1 commands, the ground station can perform the options listed in Table 2. However, after each one of the Table 2 commands, the satellite will query the ground station with a verification function. Only if the ground station sends the correct reply will the command be executed.

Table 1 - General user BBS commands. Lexical composition: in the syntax column, a capital letter represents the exact character to use, a lower case word is replaced with an appropriate response, a "#" is replaced with an integer number, elements inside brackets are optional, and elements with "|" between them means that *one,* and only one, of the elements must chosen. A "⇒" indicates that although the syntactical description continues on the next line, the second line is really a continuation of the first line.

Command	Description	Syntax
Connect	➳ Establish a communications link between a user or ground station and the satellite. ➳ Similar to logging into a computer terminal.	C PANSAT
Get current telemetry	➳ Display the telemetry values of PANSAT at that instant. ➳ Will list out formatted, with headers.	TC
Get stored telemetry	➳ Download PANSAT's telemetry for the past few days. ➳ It will download in a file. ➳ The file will be in raw data format, needing to be deciphered by the user.	TS
Send mail	➳ The command is entered on one line (excluding the message). ➳ For callsign_list, one can enter as many callsigns as desired, each separated by a space. ➳ A special callsign is "all" (meaning to every user of PANSAT). ➳ The subject is optional and denoted by beginning with a "S:". ➳ After this line is entered, beginning on the following line and going until an end of message character is reached (CTRL-D), all lines are incorporated into the message.	SM callsign_list ⇒ [S:subject] message

Command	Description	Syntax
Send file	➳ Uploads a file into PANSAT's storage. ➳ See callsign_list described in send mail. ➳ The last entry on the line is the filename, which is as it will appear on PANSAT.	SF callsign_list ⇒ filename
Read mail	➳ If a number is specified, the contents of the mail message denoted by the number is displayed onto the user's or ground station's terminal. ➳ If the 'E' option is used instead of a number, every mail messages with the callsign of the end user in the "still to" line will be sent to the end user. ➳ A user may read any mail on the system.	RM #|E
Read file	➳ Downloads the file denoted by the number. ➳ A user may download any file on the system.	RF #
Delete mail	➳ Using the 'E' option will delete every mail message that was sent to the ordinator of the command. ➳ If numbers are specified, the mail messages denoted by the number(s) are deleted. ➳ One can delete a single mail item, or a range of mail (e.g. DM 45 - 56). ➳ A user can only delete mail if that user was the originator or recipient of that particular mail. Note: deleting a mail sent to them only removes their callsign from the callsign list - once all callsigns are removed from the list, the message is removed from the system. When reading the mail, however, the original "to" list will always be displayed.	Options: DM E DM # DM # - DM - # DM # - #
Delete file	➳ The same as delete mail, except applies to files.	Same as delete mail, except use DF instead of DM. A "DF E" option is not available.
Forward mail	➳ Essentially adds new callsigns to a mail's "to" list. ➳ Eliminates the need to re-upload a mail just to increase distribution. ➳ In reading the mail, however, the forward list will be a separate line from the original "to" list. ➳ See send mail for callsign_list description.	FM callsign_list #
Forward file	➳ The same as forward mail, except applies to files.	FF callsign_list #

Command	Description	Syntax
List mail	➳ The user must pick one of four options: U - list the mail only to that particular user E - list every single mail on PANSAT N - lists out every mail on PANSAT that was stored within the previous 24 hours or every new message (no number range is allowed for this option) A - like the U option, but also include the mail sent to the "all" callsign ➳ The default is to list every mail that fits into the category, but user can specify a range (e.g. LM U 56 - means list all mail messages to that particular user beginning with and including mail number 56).	Options: LM U|E|N|A LM U|E|A # LM U|E|A # - LM U|E|A - # LM U|E|A # - #
List files	➳ The same as list mail, except applies to files.	Same as list mail, except replace LM with LF
Help	➳ Briefly describes all the available commands.	?
Switch to NPSterm	➳ Switches into using the NPSterm application layer protocol. ➳ This mode is available only to those users using NPS software to communicate with PANSAT.	NP
Switch to ASCII	➳ Switches out of the NPSterm mode.	NA
Who	➳ Lists all the users currently connected to PANSAT.	W
Send one-liner	➳ Sends the message immediately to a user currently connected to PANSAT. ➳ Can use special callsign "all" to send to every user currently connected to PANSAT. ➳ Message consists of everything after a single callsign until the end of the line.	M callsign message
Disconnect	➳ Logs off PANSAT, formally disestablishing the link.	X

Table 2 - Additional BBS commands for the Ground station only. The lexical composition is the same as in Table 1.

Command	Description	Syntax
Delete mail	➳ The same as delete mail a for general user, however no restrictions apply to deleting mail. The ground station may delete any mail message, unconditionally. ➳ The 'E' option for delete mail, listed in Table 1, is not applicable for this unconditional delete.	Options: DM # DM # - DM - # DM # - #
Delete file	➳ Same as delete mail, except applies to files.	Same as delete mail, except replace DM with DF
Post broadcast message	➳ Replaces the broadcast message which is displayed when a user first connects to PANSAT. ➳ The message works the same as in send mail.	P message
Get BBS settings	➳ The system will return a condensed block which will contain all of the current values for the user services software control settings.	G
Update BBS settings	➳ Sends a condensed data block containing all of the PANSAT user services software settings - the values sent will replaces those onboard the satellite. ➳ The ground station software automatically will generate the block from an easy-to-use interface.	U block
Update ground station callsign	➳ If the ground station will be using a different callsign in future connections, this updates that callsign so that PANSAT will be able to recognize the ground station. ➳ If a new callsign is used without updating, the satellite will only give general user privileges to the new callsign for ground station.	GS callsign
Terminate user services program	➳ Gracefully exits the user services program. ➳ All users connected to PANSAT are warned, then automatically disconnected from system. ➳ All program states are saved, dynamic memory is reclaimed, then all memory is turned over to the operating system. ➳ The user services program must be reloaded to execute again.	KI

The secondary role of the spacecraft user services software is to perform "housekeeping functions." These are experimental functions, any or all of which could be disabled or re-enabled by the ground station. Implementing these functions is a secondary priority after implementing the BBS. The "housekeeping functions" are as follows:

Deadreckoning - Periodically, the ground station will update the satellite's position. From this data, PANSAT will continually deadreckon its current position. If the satellite is in an area of no-likely communications, such as over an ocean, it will conserve power by putting the modem into a duty cycling mode. Once the period is over, the system shifts the modem back into its normal operational mode.

Fragmentation - The system will periodically check the fragmentation of the storage memory. If the fragmentation exceeds a ground station set threshold, during the next period of no-likely communications, determined using the deadreackon function detailed above, it will defragment the spacecraft's storage.

Data Compression - This will compress all the data as it is put into the storage memory on the spacecraft. This will be transparent to the user, but will maximize the available storage space.

b. Ground Station Module Functions

These functions are broken down by the specific terminal the action is performed on.

(1) Display Terminal. This terminal will have a window plotting PANSAT's current position and anticipated track onto a projection of the

world. The area along the track in which the satellite should be able to communicate with NPS will be colored. The next time the satellite is expected to be in this area will also be displayed. Another window will graphically display the spacecraft's latest telemetry. A third window will display the communications to and from PANSAT while it is in the window of reception. The final window on the display terminal will have full interaction with the PANSAT archive files. These archive files will contain telemetry and all the information from the satellite's storage that was received by the ground station. The general user's package will contain all of these windows except for the archive management window.

(2) Control Terminal. The first window on this terminal will be the communications window. This window is the means for interacting with PANSAT, as all the commands will be entered here. While the commands work as described in Tables 1 and 2, mail does not have to be written or read during the time of connection. These operations can be done offline, but only while connected to PANSAT are the mail actually transmitted and received. The operator of this terminal can enter the terminal into a "super user" mode by entering a password. While in this mode, the operator will be able to enter Table 2 commands and the terminal will automatically respond to PANSAT's verification queries. Otherwise, if in normal mode, only Table 1 commands can be entered. Another window on this terminal will be the "control panel". This window contains all the satellite's user services settings. By changing the values in this window, the spacecraft's settings will be updated at the next opportunity. On the terminal, the time until the next possible communication with PANSAT will be displayed. Another window will be an archive management window, identical to the one on the display terminal. This terminal will also have a batch job

compiler, allowing unattended interaction with PANSAT with only Table 1 commands.

(3) Windows NT Server. This will perform all the network administration. This terminal will also channel the data received from the satellite into the archives. Additionally the server will act as the Internet Virtual station. This will allow anyone connected to the Internet to access PANSAT via a World Wide Web (WWW) site. The web page will be a virtual user's HAM radio station. The functions provided by this site include the ability to post a mail message on the satellite's BBS, list the archived files, download the archived files via the standard Internet file transfer protocol, view any saved mail message or telemetry data, or request a mail message be downloaded for viewing during the next pass. An Internet viewer could see a desired mail message in the mail listings, which are downloaded with every pass. However, the actual mail may not be yet downloaded. Thus this viewer would need to request the mail for downloading during the next pass. After each pass, these Internet pages are regenerated with the latest data. Implementing the Internet virtual station is a secondary requirement after the development of the control and display terminals.

(4) Communication Terminal/Linux Box. This computer serves as the communication relay between the ground station and the satellite. It will act as the TNC, controlling the serial communication controller (SCC) chip and modem. Packets of data received from PANSAT are relayed to the three NT terminals, while data received from any of the terminals is transmitted to the satellite. All the packets transmitted or received are locally backed up for twenty-four hours.

3. User Characteristics

While a level of competence is not assumed, both types of operators of the system (user and ground station) are expected to be fairly knowledgeable with HAM radio operations, with at least some familiarity with satellite communications and a BBS.

4. General Constraints

a. Timing Constraints

PANSAT is a soft real-time system. That is, the timing of the information flow does not effect the integrity of the system. A piece of information that arrives slowly has no adverse impact on the system, other than just taking a long time. However, because this is a communications platform with a limited window of opportunity (two to ten minutes per pass, a few passes per day), the system is better utilized by speedy operations. To support optimized communications, the onboard satellite processing and system overhead needs to appear to the end user to take a nearly no processing time. That is, all the window of opportunity to communicate should not be "wasted" waiting on processing. Thus for the spacecraft module, the highest priority is code speed coupled with a small program size. The ground station module's highest priority is simply code speed, since program size is not a relative factor.

b. Spacecraft Module Constraints

This module will run on Intel's 80C186 processor, executing on top of BekTek's spacecraft operating system (SCOS) and utilizing BekTek's AX.25 protocol utilities. The interfaces to both of BekTek's units are via Microsoft C version 5.0/5.1 object files. The user services program will have about 300 kB out of 512 kB of memory left over to work with (combined code and data), but the goal is to have the program use as little memory as possible. For storage space, PANSAT will use two 4.5 MB (4 MB of normal RAM, 0.5 MB of FLASH ROM)

memory banks. Initially the two memory banks will be mirroring each other for backup redundancy. However, if single-bank reliability experiments prove successful, the banks will be used in conjunction, providing 9 MB of storage.

c. Ground Station Module Constraints

This module will run on two PCs. Initially, the software will execute on top of the Windows 95/NT operating system. Follow-on projects, however, will port the program over to DOS-based and Linux-based systems. The purpose of these ported versions is to offer diversity in the general user's package. However, these are secondary and tertiary requirements, respectively. Thus, for ease of portability, operating system dependent code needs to be localized as much as possible.

5. Assumptions

For the spacecraft module, the operating system, packet protocol, log and telemetry management utilities will be integrated with the user services software by the SSAG. These utilities provide all the hardware interfaces required by the user services program. Likewise, the hardware interfaces for the ground station will also be provided by the SSAG. This includes drivers to control the modem, the SCC, the antenna rotor, and AX.25 protocol utilities.

C. SPECIFIC REQUIREMENTS

This section breaks down the general descriptions provided in section B.2 into specific individual functional elements.

1. Spacecraft Module

a. BBS Functions

(1) Login User / Ground Station

Introduction - PANSAT will normally be in a receive-mode, waiting for a request-to-connect command from a user. Even when a connection is ongoing, the satellite is concurrently waiting for another user to connect, until the maximum number of connections has been made.

Inputs - The connection process is triggered by the input stream "C PANSAT" from a user or the ground station received via SCOS. The identity/origin of the data stream, which is determined by the callsign of the user sending the stream, will be provided by the AX.25 packet utilities.

Processing - If one of the fifteen connection records are available, PANSAT will modify the available connection record from "inactive" to "active". If the identity of the new connection is the ground station, then the satellite will activate the reserved ground station connection record instead. The time and date that the ground station is connecting will be sent to the log manager, as well as unsuccessful ground station login attempts. After the connection record is updated, control for this record is passed to the function in next section. If no records are available, the user is notified of the failed connection, and no control is not passed. If the "terminate user services program" process is in progress, which

means the program is shutting down, all connection requests will be denied. While control for a particular connection record may pass to another function, this function will continue to process, waiting for another connection request.

Outputs - The connection records are updated - the record is marked as "active", the callsign is stored in record, the terminal mode is set to ASCII, and the connection status is set to "connecting". The same occurs for the ground station, except callsign is not stored in the connection record, because it is already stored there. If no connections are available, the "Sorry, no connections available" message stream is sent to the attempting callsign via SCOS. The ground station is guaranteed a reserved connection, thus would never get this message. If the "terminate user services program" process is in progress, the message "System is temporarily unavailable" is sent to the user.

External Interfaces - *Note that before data is sent to SCOS for transmission, it must first be sent to the AX.25 utilities for proper packeting. Likewise any packet received from SCOS must be sent to the AX.25 utilities to be deciphered/made readable. For sake of space, this fact will not be repeated in further interface sections.* Interfacing in this function will be done via those two packages (SCOS and the AX.25 utilities). Additionally, if the ground station logs in, the log manager package will also be interfaced.

Performance Requirements - This function needs to be continually running. When this function passes control to anther

function, it is for that connection or callsign only. This function continues to execute, waiting for another new connection.

Design Constraints - The best way to implement this function will be to have an incoming stream parser. If a stream is from an ongoing connection, it sends the data to the appropriate function, based on the connection status field. Otherwise, the data stream is sent to this function. Every time an active connection receives data from the user, the "time of last input from user" field in the connection record will be updated.

Attributes:

Ground Station Identity: The initial ground station callsign will be "KD6CXV". This can be changed by the ground station.

Maximum Connections: Maximum number of connections will be 15 concurrently interleaved users, with an additional connection reserved for the ground station.

Other Requirements - None.

(2) Send Greeting to New Connection

Introduction - Once a connection has been established, a greeting message followed by a happy face is sent to the user or ground station.

Inputs - None.

Processing - The stream "Welcome to PANSAT" is sent to the connection. If the connection is the ground station, also send "The last communication with the ground station was ..." followed by the "time of last input from user" field in the ground station's connection record. For any type connection, the greeting is ended with a happy face. The happy face will be made out of selected current telemetry values, the specifics of which will be determined later.

Outputs - The canned greeting statements are sent to the connection via SCOS.

External Interfaces - All interfacing in this function is done via the SCOS, AX.25, and telemetry manager utility packages.

Performance Requirements - The happy face should not be very complicated.

Design Constraints - The happy face only needs to contain a few simple telemetry values, just to quickly let the knowledgeable user view PANSAT's status.

Attributes - None.

Other Requirements - None.

(3) Check File Transfer Status

Introduction - When a new connection has been established, this function compares the callsign to a list of callsigns that were previously disconnected in the middle of a file transfer. If the new callsign matches one on the list, the user will be asked if the file transfer is to continue. If affirmative, the file transfer continues exactly where it left off..

Inputs - The answer whether to continue the file transfer or not is received from the user via SCOS.

Processing - The connecting callsign is compared to the list of interrupted file transfer users. If a match exists, the user is queried if the file transfer is to continue. If yes and the file transfer was an upload, the status is set to "temp uploading"and the file on PANSAT is opened for append, then control is passed to the file upload function. If the file transfer was a download, the status is set to "temp downloading" and the position in the file is retrieved, then the file download function is called, beginning at the position indicated. After the file transfer is complete, control will return to this function and the callsign will be removed from the list. Additionally, if the connecting callsign was in the interrupted file transfer list, but the user chose not to continue the operation, the callsign will be removed from the list. When the file transfer is done or if the operation was skipped, control is passed to function in the next section.

Outputs - The question "Do you want to continue the file transfer?" is sent to the end user via SCOS. If an upload, an append command is sent to the file system, otherwise the file is opened and the file position pointer is set. The file transfer function then handles the rest of the input/output.

External Interfaces - All interfacing in this function is done via the SCOS, with the Surrey file operations application, and AX.25 utility packages.

Performance Requirements - None.

Design Constraints - The connection status is set to "temp download" or "temp upload", rather than the normal download or uploading status. That way, after the file transfer is complete, the program knows to return to the connection process, where it continues where it left off.

Attributes - In the file-transfer interrupted list, the following values need to be stored: the callsign, date-time stamp of the disconnect, whether uploading or downloading, the filename accessing, and the position in the file.

Other Requirements - None.

(4) Broadcast Message

Introduction - The last step in the connection status is to send the user the broadcast message. This message is generated solely by

the ground station, but is displayed to every user connecting to PANSAT.

Inputs - The broadcast message is retrieved from the file system (the Surrey part of SCOS).

Processing - First the broadcast message is relayed to the end user. Additionally, at this time the "time of last input from user" field in the connection record is updated with the current time and the connection status is set to "menu". The connection process is now completed and control is passed to the function in next section.

Outputs - The broadcast message is sent to the user via SCOS. The connection record is updated.

External Interfaces - All interfacing in this function is done via the SCOS, with the Surrey file operations application, and AX.25 utility packages.

Performance Requirements - None.

Design Constraints - This is the last part of the connection process.

Attributes - None.

Other Requirements - None.

(5) Options Menu. When the connection is in the menu state, the system sends a line with all the possible letter choices to the end user (i.e. TC, TS, SM, SF, RM, RF, DM, DF, LM, LF, FM, FF, NP, NA, NP, W, M, ?, X >), then waits for the user or ground station to send a command. If the command sent by the user does not fit the format described in Tables 1 and 2, the end user is sent a warning. By receiving the command prompt, the end user knows the previous command is completed and PANSAT is ready for the next command. This function will repeat until the user enters the disconnect command or is automatically disconnected by the system.

(a) Get Current Telemetry

Introduction - See Table 1.

Inputs - This function is triggered by the user or ground station sending a stream with the command "TC". Current telemetry values are obtained from the telemetry handler.

Processing - The telemetry values are inserted into a preformatted mask, then sent to the user or ground station. Control is returned to the menu after completion.

Outputs - The filled-out mask is sent to the user via data streams in SCOS.

External Interfaces - All interfacing in this function is done via the SCOS, the telemetry handler, and the AX.25 utility packages.

Performance Requirements - The telemetry values must be current. Turn around for the information must be quick enough so a time lag is not noticeable.

Design Constraints - Since this is a one step operation, the connection status stays as "menu".

Attributes - None.

Other Requirements - The actual values of the telemetry have not yet been defined by the client. Additionally, the means of accessing the current telemetry values has not been provided. Both of these need to be completed before this function can be implemented.

(b) Get Stored Telemetry

Introduction - See Table 1.

Inputs - This function is triggered by the user or ground station sending a stream with the command "TS".

Processing - The file download function is called with the identity of the stored telemetry file. When the file transfer is completed, control loops back to menu function.

Outputs - The connection status is set to "downloading". After completing the download, the status is set back to

"menu". The actual file to download is directed from the file system to the user via SCOS.

External Interfaces - All interfacing in this function is done via the SCOS, the Scurry file system, and the AX.25 utility packages.

Performance Requirements - None.

Design Constraints - None.

Attributes - The filename of the stored telemetry will be a hard coded program constant.

Other Requirements - The client still needs to provide the name of the file in which the telemetry will be stored.

(c) Send Mail

Introduction - See Table 1.

Inputs - This function is triggered by the user or ground station sending a stream with the command "SM". The mail input format is described in the send mail syntax of Table 1. These inputs are received from the end user via SCOS.

Processing - After the send mail command is received, all input from the user until a end-of-message character is considered part of the mail message. In saving the message,

a file is created and a "to" line, a "from" line, and a "subject" line are added at the beginning of the text. Additionally, at the very beginning of the file a "still to" line is added. While this line is not viewed by the user, it is used by the system to tell who in the "to" line has yet to view the message. Initially, the "still to" will be a duplicate of the "to" line. If the restrict mail/file size setting is "ON" and the mail exceeds the threshold size, or if there is not enough room in storage to save the mail, the mail will not be saved and a warning will be sent to the user. When the end-of-message character is received, the file containing the mail message is closed.

Outputs - After the SM line is received, the connection status is set to "sending mail". After the end-of-message character is received, the status is set back to "menu". If the sending mail is canceled due to a mail or storage size error, the user is notified that the "Mail size exceeds system maximum, not saved".

External Interfaces - All interfacing in this function is done via the SCOS, with the Surrey file operations application, and AX.25 utility packages.

Performance Requirements - None.

Design Constraints - All mail is assumed to be in text/ASCII format.

Attributes - The end-of-message character is a CTRL-D.

Other Requirements - None.

(d) Send File

Introduction - See Table 1.

Inputs - This function is triggered by the user or ground station sending a stream with the command "SF". The format for the file input is described in the send file syntax of Table 1. All input after the "SF" line until an end-of-file is reached will be considered part of the file. These inputs are received from the end user via SCOS.

Processing - The system attempts to create a file. Each file is identified by a user service's number, thus filenames on the system can be redundant. If the file is already opened in append mode by the interrupted file transfer continuation process, which is described in "check file transfer status" section above, use that opened file. Save "to", "from", and "still to" information is a separate file. This file information file will have the same name as the actual file, except the first character of filename extension will be replaced with a "!". This process is skipped for the appended file, as the information is already there. The connection status is set to "uploading" and all data from the user until the end-of-file is added to the file. After the file transfer has been completed, the file is closed. If the restrict mail/file size setting is "ON"

and the file exceeds the threshold size, or if there is not enough room in storage to save the file, do not save the file and send a warning to the end user. After the file transfer is completed, the connection status is returned to "menu".

Outputs - If the filename already exists on the system, send a warning to the user. After the file transfer is completed, notify the user that transfer is complete and how much storage space was taken up by the file. During a file transfer, send all the file data to the storage system, via the Surrey file system. When transfer begins, set connection status mode to "uploading". If the upload is canceled due to a file or storage size error, notify user "File size exceeds system maximum, not saved".

External Interfaces - All interfacing in this function is done via the SCOS, with the Surrey file operations application, and AX.25 utility packages.

Performance Requirements - A file transfer protocol similar to the PACSAT level 0 ftp will be implemented for PANSAT.

Design Constraints - Unlike mail, there is no assumption on the format of a file. It is treated as a binary file.

Attributes - None.

Other Requirements - None.

(e) Read Mail

Introduction - See Table 1.

Inputs - This function is triggered by the user or ground station sending a stream with the command "RM". The read mail input format is described in Table 1. This input is received via SCOS.

Processing - If the mail identified by # exists, then the file holding the mail message is retrieved and sent to the user. If # does not exist, a warning is sent to the user. If the 'E' option was used instead of a number, every mail message with the end user's callsign in the "still to" line will be sent to end user in one continuous stream. When the operation is done, control is passed to the menu.

Outputs - Simply dumps the contents of the mail to the end user. However, the "still to" line is not relayed to the user or ground station. If mail does not exist, the warning "Mail # does not exist" is sent to the user or ground station. If the 'E' option was used, but no mail for the end user exists, the warning "No mail for callsign" is sent to the end user.

External Interfaces - All interfacing in this function is done via the SCOS, with the Surrey file operations application, and AX.25 utility packages.

Performance Requirements - None.

Design Constraints - This function does not need to modify connection status.

Attributes - None.

Other Requirements - None.

(f) Read File

Introduction - See Table 1.

Inputs - This function is triggered by the user or ground station sending a stream with the command "RF". The read file input format is described in Table 1. This input is received via SCOS.

Processing - If file # exists, the size is sent to the user. The file is then opened and read until end of file, relaying the data to user. The position in file is continually updated in connection record. If file # doesn't exist, the user is sent a warning message. However, this warning is prefaced with an end-of-file signal, so that the end user will exit their downloading mode. When the file transfer is done, control is passed to the menu function.

Outputs - The connection mode is set to "downloading". When complete, the status is reset to "menu". The file contents are sent to the user via SCOS. If the file #

specified by the user does not exist, the warning "File # does not exist, operation aborted" is sent to the user.

External Interfaces - All interfacing in this function is done via the SCOS, with the Surrey file operations application, and AX.25 utility packages.

Performance Requirements - None.

Design Constraints - A file transfer protocol similar to the PACSAT level 0 ftp will be implemented for PANSAT.

Attributes - None.

Other Requirements - None.

(g) Delete Mail

Introduction - See Tables 1 and 2. Note the possible combinations of the command line, for example: "DM 43" deletes one mail; "DM - 43" deletes all mail up to and including mail number 43; "DM 43 - 56" deletes all mail between and including 43 and 56; and "DM 43 -" deletes all mail with a number equal to or greater than 43.

Inputs - This function is triggered by the user or ground station sending a stream with the command "DM". The delete mail input format is described in Table 1 or 2. This input is received via SCOS.

Processing - A user may only delete mail if that user was the originator or recipient of that particular mail. If the originator deletes it, it is immediately removed from the system. However, if user who received the mail deletes it, it really only deletes that user's callsign from the "still to" list. If the "still to" list becomes empty, then the mail is removed from the system. This way the users still on the "still to" list can read the mail before it is removed. If the # specified is unable to be deleted or does not exist, the user is warned, then the process continues by deleting what files it can from those specified in the command line. If the 'E' option is used instead of a number range, mail message with the end user's callsign in the "still to" line will be deleted, as described above. After this function completes, control is returned to menu. The ground station will normally use the delete mail command as a regular user. However, the ground station can purge any mail from the system by appending a "SU" at the end of the delete command. This means super user delete or unconditional removal of all files specified. The command line is put into a special command buffer and control is passed to the verification function, specified in its section below. After the ground station is verified, every file specified is removed from the system.

Outputs - The user or ground station is notified if a number specified (or a number in the range they specified) is unable to be deleted, due to the # not existing or the user not having privileges to delete it. If the 'E' option was used, but no mails exist with the end user's callsign in the "still to"

line, the warning "No mails to delete" is sent to the end user. Otherwise, the end user is notified upon the successful completion of the delete operation.

External Interfaces - All interfacing in this function is done via the SCOS, with the Surrey file operations application, and AX.25 utility packages.

Performance Requirements - None.

Design Constraints - This function does not require any connection status changes.

Attributes - None.

Other Requirements - None.

(h) Delete File. This function works the exact same way as delete mail, except it applies to files and is triggered by a "DF" command.

(i) List Mail

Introduction - See Table 1.

Inputs - This function is triggered by the user or ground station sending a stream with the command "LM". The list mail input format is described in Table 1. This input is received via SCOS.

Processing - The directory structure is simply accessed from file system and the file numbers matching the range specified in the input command are kept while the others are discarded. See Table 1 for the matching criteria. These matches are sent to the user. If the "N" option is used, however, instead of mail numbers, the date the mail was posted is checked. If the age of the mail is equal to or less than 24 hours, it is listed to the user, otherwise it is ignored.

Outputs - Matching directory entries are sent in the following format, prefaced with this heading:
Mail # To From Date/Time Subject

External Interfaces - All interfacing in this function is done via the SCOS, with the Surrey file operations application, and AX.25 utility packages.

Performance Requirements - None.

Design Constraints - This function does not require a connection status change.

Attributes - None.

Other Requirements - None.

(j) List File. Works the same as list mail except applies to files, is triggered by the "LF" command, and is in the following format:

File # To From Date/Time Filename

(k) Who

Introduction - See Table 1.

Inputs - This function is triggered by the user or ground station sending a stream with the command "W". This input is received via SCOS.

Processing - A list of all the callsigns from the active connection records is sent to the end user via SCOS.

Outputs - A data stream containing all the callsigns currently connected on the system is sent to the user.

External Interfaces - All interfacing in this function is done via the SCOS and AX.25 utility packages.

Performance Requirements - None.

Design Constraints - To obtain the callsigns currently connected to PANSAT, the system only needs to examine the active connection records. There is no need to change connection status for this function.

Attributes - None.

Other Requirements - None.

(l) Help

Introduction - See Table 1.

Inputs - This function is triggered by the user sending a stream with the command "?". This input is received via SCOS.

Processing - Whenever this command is received, PANSAT simply outputs the help table, then returns to the menu.

Outputs - In the format of table 1, the Command Name and Syntax columns are listed out to the user via SCOS.

External Interfaces - All interfacing in this function is done via the SCOS and AX.25 utility packages.

Performance Requirements - None.

Design Constraints - No change in connection status is required.

Attributes - None.

Other Requirements - This command is performed on ASCII terminals only. If the connection is using NPSterm or NPS software, the command is intercepted by the host terminal and given a much more robust response.

(m) Forward Mail

Introduction - See Table 1.

Inputs - This function is triggered by the user or ground station sending a stream with the command "FM". The forward mail input format is described in Table 1. This input is received via SCOS.

Processing - If the mail # exists, a "Forward" line is added to the top of the mail file and the callsign_list is concatenated to the "still to" field. If the mail # does not exist, the end user is warned.

Outputs - If mail # does not exist, the warning "Mail # does not exist" is sent to the end user.

External Interfaces - All interfacing in this function is done via the SCOS, with the Surrey file operations application, and AX.25 utility packages.

Performance Requirements - None.

Design Constraints - No change in the connection status is required for this function.

Attributes - None.

Other Requirements - None.

(n) Forward File. The function works the same as forward mail, except it applies to files and is triggered by the "FF" command.

(o) Switch to NPSterm

Introduction - See Table 1. This application-level protocol uses two methods to cut the interaction between the end user and the satellite. The basic principle for the reduced interaction is the less data transferred, the less time is used conducting the interaction, making the limited window of communication with the satellite more productive. Shortcuts are used for standard commands and canned statements, and all information sent between the user and PANSAT is applied through a data compression scheme. The version of NPSterm is sent to the user/ground station to ensure that the sender and receiver are compatible.

Inputs - This function is triggered by the user or ground station sending a stream with the command "NP". This input is received via SCOS.

Processing - The connection record is updated to indicate the end user is using NPSterm.

Outputs - A quick statement is sent to the user notifying that NPSterm is about to be entered. Additionally, the version of NPSterm used by PANSAT is sent to the user. Both messages are sent via SCOS.

External Interfaces - All interfacing in this function is done via the SCOS and AX.25 utility packages.

Performance Requirements - It is desired to use a compression routine that is optimal towards text, as this is anticipated to encompass the bulk of communications for PANSAT.

Design Constraints - All interaction between the end user and PANSAT after this command is executed will use shortcuts and data compression.

Attributes - The version of NPSterm will be a hard coded program constant.

Other Requirements - The listing of the NPSterm shortcuts has yet to be defined, so, for an initial version of the protocol, only the data compression will be implemented.

(p) Switch to ASCII

Introduction - See Table 1. Drops the connection out of NPSterm mode.

Inputs - This function is triggered by the user or ground station sending a stream with the command "NA". This input is received via SCOS.

Processing - The connection record is updated to indicate that the end user is using an ASCII terminal interface.

Outputs - Sends a quick statement that NPSterm is about to be exited to the user via SCOS.

External Interfaces - All interfacing in this function is done via the SCOS and AX.25 utility packages.

Performance Requirements - None.

Design Constraints - None.

Attributes - None.

Other Requirements - None.

(q) Send One-Liner

Introduction - See Table 1.

is over. For NPSterm, however, all processes can be interrupted, even file transfers, with no harm.

Attributes - None.

Other Requirements - None.

(r) Disconnect

Introduction - See Table 1.

Inputs - This function is triggered by the user or ground station sending a stream with the command "X". This input is received via SCOS.

Processing - Reclaims the connection record used by that callsign. Marks the record as "inactive".

Outputs - Sends "Disconnected from PANSAT" message to the end user just before disconnecting.

External Interfaces - All interfacing in this function is done via the SCOS and AX.25 utility packages.

Performance Requirements - None.

Design Constraints - None.

Attributes - None.

Inputs - This function is triggered by the user or ground station sending a stream with the command "M". The send one-liner input format is described in Table 1. This input is received via SCOS.

Processing - If in the callsign position the special callsign "all" is used, then the message is sent to every end user currently connected to PANSAT. Otherwise, after checking to ensure the callsign is currently connected to PANSAT, the message is immediately sent to the callsign. If the callsign is not currently connected, a warning is sent to the originator.

Outputs - If the callsign is connected to PANSAT, the message is sent to them. Otherwise the warning "Callsign not connected to PANSAT" is sent to the originating user.

External Interfaces - All interfacing in this function is done via the SCOS and AX.25 utility packages.

Performance Requirements - While the line should get sent to the end user instantly, in no way should that actually interfere with the end user's current activity.

Design Constraints - For an ASCII terminal, this operation can interrupt any operation without disrupting it, with the exception of file transfers. Thus, for ASCII terminals, the sending of the message will be delayed until the file transfer

Other Requirements - None.

(s) Post Broadcast Message

Introduction - See Table 2. This is a ground station only function.

Inputs - This function is triggered by the ground station sending a stream with the command "P". The broadcast message input format is described in the post broadcast message syntax of Table 2. These inputs are received from the user via SCOS.

Processing - The command is placed in the special command buffer, then control is passed to the verification function. After the verification is authenticated, the message received from the input overwrites the file containing the old broadcast message. The message is updated similarly to uploading a mail message (see send mail section above).

Outputs - A message is sent to ground station acknowledging that the broadcast message update has been sent, via SCOS. A file containing the broadcast message is sent to file system for storage. After the "P" line and verification have been received, the connection status is set to "sending broadcast". After the end-of-message character is received, the status is set back to "menu".

External Interfaces - All interfacing in this function is done via the SCOS, with the Surrey file operations application, and AX.25 utility packages.

Performance Requirements - None.

Design Constraints - There can only be one broadcast message, so by posting a new message, the old message is removed. If the old message is desired to be kept in addition to a new message, the two messages must concatenated by the ground station, then presented to PANSAT as a single broadcast message.

Attributes - The end-of-message character is a CTRL-D.

Other Requirements - None.

(t) Get BBS Settings

Introduction - See Table 2. This is a ground station only function.

Inputs - This function is triggered by the ground station sending a stream with the command "G". This input is received via SCOS.

Processing - The command is placed in the special command buffer and sent to the verification function. After the verification has been authenticated, PANSAT will send

all the control settings to the ground station in a concentrated block.

Outputs - The settings are listed in a single concatenated string and made up of the following values (Table 3), in the order listed.

Table 3 - User services control settings, with default values.

Setting Description	Setting Type	Default Value
Use the data compression algorithm on all data saved to the satellite's storage?	Boolean	FALSE
Deadreckon the satellite's position from an initial position provided by the ground station? If so, put the modem in a duty-cycling mode if PANSAT is above areas with no-likely communication.	Boolean	FALSE
Conduct defragmenting of the satellite's storage during periods of no-likely communications? (This value can only be TRUE if the deadreckoning flag is also TRUE)	Boolean	FALSE
The percentage of fragmentation in the storage space below which to begin the defragmentation process. (This is ignored if the defragmenting flag is FALSE)	Integer	85
Check the signal strength of all received packets? If so and the signal is above a threshold set in software, warn the connection. If three more packets are received at the same high power, then disconnect the user.	Boolean	FALSE
Number of minutes after no communication with an active connection to automatically disconnect it.	Integer	2
Restrict the size of stored mail message and files?	Boolean	TRUE
The maximum mail size to allow in storage, in kB. (This is ignored if the restrict size flag is FALSE)	Integer	4
The maximum file size to allow in storage, in kB. (This is ignored if the restrict size flag is FALSE)	Integer	256

Setting Description	Setting Type	Default Value
The maximum number of files sent to "all" to allow on the system at any one time.	Integer	25
The maximum number of connection records to store for users who were disconnected during a file transfer.	Integer	10
The maximum number of days to store connection records for users who were disconnected during a file transfer.	Integer	3
The number of days after posting in which normal mail or files (those not sent to "all") should always be deleted.	Integer	14
The number of days after posting in which mail or files sent to "all" should always be deleted.	Integer	7
Perform the additional autodelete functions (listed below)?	Boolean	TRUE
If the additional autodelete function is enabled (TRUE), this is the threshold amount of memory left in storage, in kB, to begin the autodelete process. Uses the next three settings as the conditional values in determining the autodelete.	Integer	500
Number of kB and days, respectively, to autodelete files. This means that only if a file is larger than the size indicated and older than the age threshold, the file is deleted.	Integer, Integer	40, 4
Second kB/days autodelete threshold.	Integer, Integer	30, 5
Third kB/days autodelete threshold.	Integer, Integer	20, 6
The current time and date to set PANSAT's clock.	Time/date stamp	N/A
The time and date of the calculated satellite orbital elements (the next six table entries)	Time/date stamp	N/A
Inclination (degrees) *	Real	N/A
Right ascension of ascending node (degrees) *	Real	N/A
Eccentricity *	Real	N/A
Argument of Perigee (degrees) *	Real	N/A
Mean Anomaly (degrees) *	Real	N/A

Setting Description	Setting Type	Default Value
Mean Motion (revolutions/day) *	Real	N/A

* These data elements provide a calculated position and track on which the deadreckoning functions calculations are based.

External Interfaces - All interfacing in this function is done via the SCOS and AX.25 utility packages.

Performance Requirements - None.

Design Constraints - The block of information is sent in duplicate, so the ground station can detect if an error occurred in the transmission. This extra reliability is added since the ground station knowing PANSAT's control settings can be critical. No change in connection status is required for this function.

Attributes - None.

Other Requirements - None.

(u) Update BBS Settings

Introduction - See Table 2. This is a ground station only function.

Inputs - This function is triggered by the ground station sending a stream with the command "U". The update

settings format is described in Table 2. The format of the information is provided in Table 3. These inputs are received from the user via SCOS.

Processing - The command line is put into the special command buffer, then control is passed to the verification function. After the verification has been authenticated, the block of values are broken out and replace all of the current control settings onboard the satellite.

Outputs - If settings are accepted, a message is sent to ground station confirming the change. Otherwise the ground station is notified of the need to retransmit the settings.

External Interfaces - All interfacing in this function is done via the SCOS and AX.25 utility packages.

Performance Requirements - None.

Design Constraints - The settings are sent in duplicate. That way, if the two settings do not match, PANSAT knows there was an error in the transmission and the settings will be rejected. This is done since getting the correct value for the settings is highly important to satellite operations. No change in connection status is required for this function.

Attributes - None.

Other Requirements - None.

(v) Update Ground Station Callsign

Introduction - See table 2. This is a ground station only function.

Inputs - This function is triggered by the ground station sending a stream with the command "GS". The update ground station callsign format is described in Table 2. These inputs are received from the user via SCOS.

Processing - The command line is placed inside the special command buffer, then control is passed to the verification function. If the verification is authenticated, the ground station callsign is replaced with the one in the command line.

Outputs - A statement saying whether the ground station callsign has been changed or not is sent to the ground station, via SCOS.

External Interfaces - All interfacing in this function is done via the SCOS and AX.25 utility packages.

Performance Requirements - None.

Design Constraints - The new callsign will be in effect the next time the ground station connects. The new callsign

will be sent in triplicate in the command line. If there is any difference in the three callsigns, the entire operation is aborted and a message is sent to the ground station. Getting the callsign correct is critical since any further connections by the ground station would not be recognized with a wrong callsign. This would effectively prohibit the ground station from ever performing Table 2 commands again, until the satellite is reset.

Attributes - The ground station callsign is stored in the connection record reserved for the ground station.

Other Requirements - The initial ground station callsign will be "KD6CXV".

(w) Terminate User Services Program

Introduction - See Table 2. This is a ground station only function. This command would only typically be used to allow an updated version of the user services program to be loaded on PANSAT, then have operations continue where they left off.

Inputs - This function is triggered by the ground station sending a stream with the command "KI". This input is received via SCOS.

Processing - The command is placed in the special command buffer and sent to the verification function. After

the verification has been authenticated, all connection requests received are denied and all users currently connected to PANSAT are sent a warning. After waiting for one minute, all users still connected to PANSAT will be automatically disconnected. All open files will then be closed, all dynamic memory will be deallocated, and the program will terminate.

Outputs - The message "System will be temporarily shut down in one minute - please disconnect now" is sent to the users connected to PANSAT.

External Interfaces - All interfacing in this function is done via the SCOS, with the Surrey file operations application, and AX.25 utility packages.

Performance Requirements - After the warning to disconnect is delivered, a full minute is allowed before automatic disconnection occurs. This allows the users to finish their business at hand before being "kicked off" the system.

Design Constraints - If a user is in the middle of a file transfer when the terminate command is received, the file transfer process must be terminated. If the file transfer was not interrupted, the user would never receive the disconnect warning, and thus would not understand what happened when the disconnection occurred.

Attributes - None.

Other Requirements - None.

(x) Verification Function

Introduction - This is not a menu option, but rather a function called by certain menu options. Whenever a command specified in Table 2 is entered, PANSAT first checks to make sure the command is being issued by the ground station. If it is, control is passed into this function to ensure the identity is the true ground station. This is done by sending a short series of numbers to the alleged ground station. If the next communication from the ground station is the correct answer to the numbers based on the application of a mathematical formula, the command located in the special command buffer is executed.

Inputs - From the alleged ground station, the data stream containing the an answer is obtained via SCOS.

Processing - Once this function is called, it creates a short stream of numbers via random number generation. It sends the numbers to the ground station and compares the answer received to the internally calculated answer. If the answer is correct, the command located in the special command buffer is executed, according to the specifications listed in that commands respective section. Otherwise, the date and time

of the error is sent to the log manager and the special command buffer is ignored.

Outputs - A short string of numbers is sent to the ground station via SCOS. If a wrong answer is received, the warning "Incorrect verification function, command not executed" is sent to the supposed ground station via SCOS. Further, the date/time stamp is sent to the log manager utility.

External Interfaces - All interfacing in this function is done via the SCOS, the log manager, and AX.25 utility packages.

Performance Requirements - To minimize errors, the stream will be sent in triplicate, one complete stream followed by another. That way, if a couple of numbers are damaged, a "best-two-out-of-three rule" should be able to provide the correct string of numbers. Likewise, the answer should be sent in triplicate as well. If two out of the three answers are correct, the answer is correct. However, to minimize overhead and time, the stream should fit into a single AX.25 data packet.

Design Constraints - Integer/binary math should be used in the verify functions in order to keep the processor load to a minimum, since all floating point operations are emulated by the processor via software.

Attributes - For security, the actual function will not be published, but will be handed over to ground station personnel at the time of software turnover.

Other Requirements - None.

(6) Autodelete

Introduction - This function is automatically run by the system. It deletes old files and mail messages from the system, so that storage does not get used up by old data. At the same time, all connections stored in the interrupted file transfer list are checked for purging.

Inputs - The size and date/time stamp of every mail and file are obtained from the file system.

Processing - This will compare the age, which is determined by current date minus the file date, of every mail and file to the delete file settings. There is one setting for the mail/files sent to "all", another setting for all other mail/files. If the mail or file is older than the setting, it will remove it from the system. Further, if the additional delete setting is "ON" and the amount of free memory remaining in storage is less than or equal to the threshold set, it will activate the second kind of autodelete. There can be up to three settings (see Table 3), each one specifying a file size and age. If a file is equal to or larger than the specified size and older than the specified age, it will be removed from the system. This repeats for each one of the three settings. Additionally, the age of each interrupted file transfer connection record saved is compared to the

threshold age set in the control block. If the connection record is older than the age, it is purged from the list.

Outputs - None.

External Interfaces - All interfacing in this function is done via the Surrey file operations application utility.

Performance Requirements - This function is automatically triggered once per day.

Design Constraints - This is a concurrent task to the BBS functions.

Attributes - None.

Other Requirements - None.

(7) Autodisconnect

Introduction - If no input stream is received from a connection in a threshold set number of minutes, the user is automatically disconnected and the connection record is recycled for another user.

Inputs - None.

Processing - Every active connection's record time stamp is compared with the current time. If difference in time is greater than

the threshold set by the ground station, the user is disconnected. If the connection status was either "uploading" or "downloading", the connection record information will be saved in the file-transfer interrupted list, with the values as specified in attribute section of the "check file transfer status" description above. If the status was either "temp uploading" or "temp downloading", the connection record already stored in the file-transfer interrupted list is merely updated.

Outputs - None.

External Interfaces - No interfaces are used in this function.

Performance Requirements - The connections should be checked once per minute.

Design Constraints - Every time a data stream arrives from a connection, that connection record's "time of last input from user" field will be updated with the current time. This function needs to be a concurrent task from the BBS functions.

Attributes - The number of minutes to autodisconnect a connection is set in the control settings block.

Other Requirements - None.

b. Housekeeping Functions

(1) Deadreckoning Function

Introduction - PANSAT determines its current position based on an initial position provided by the ground station and time passed. Using this position, the satellite will determine when it is in periods of no- likely communications and switch the modem into an energy saving mode during these times.

Inputs - None.

Processing - Using the satellite position initialized by the ground station (it needs to be periodically updated by the ground station), PANSAT deadreckons its current position. It will then compare its current position with areas known to have no-likely communications. When it enters or exits these areas, the defragmenting function (next section) will be notified. When it enters the area, the modem will be shift into a duty-cycling mode to conserve power. The modem will be switched back to its normal mode upon exiting the area.

Outputs - A signal is sent to the modem in order to switch it into and out of a duty-cycling mode

External Interfaces - The only interface used is the driver for the modem.

Performance Requirements - The satellite should deadreckon its position once per minute.

Design Constraints - This function needs to be a concurrent task.

Attributes - Areas of no-likely communications will be hard coded program constants.

Other Requirements - If no contact with the ground station is made within 24 hours, this feature is automatically disabled. To determine the time of the last ground station contact, the current date and time is compared to the date/time stamp stored in the "time of last input from user" field of the ground station's reserved connection record.

(2) Defragment Function

Introduction - If the defragmentation flag is set "ON", PANSAT periodically checks the fragmentation of the storage system. If fragmentation is below a threshold and the satellite is in a period of no-likely communications, the storage is defragmented.

Inputs - None.

Processing - A period of no-likely communications is determined in the previous section. Once the period is entered, a new fragmentation value is determined. If the defragmentation setting is ON, as many files as possible until the period of no-likely communications is over are defragmented.

Outputs - None, an unfragmented storage space is the result.

External Interfaces - The SCOS with the Surrey file system is the only interface used by this function.

Performance Requirements - Updating the fragmentation value should be done at the beginning of every no-likely communications period. Defragmentation needs to be a quick, interruptible process since the period of no-likely communications will be a short time.

Design Constraints - Defragmentation should be performed one file at a time to allow maximum flexibility.

Attributes - The fragmentation threshold is obtained from BBS control block settings.

Other Requirements - This assumes the fragmentation value can be obtained from the file operating system, which has not yet been tested.

(3) Data Compression Functions

(a) Compress Data

Introduction - If the compression flag is set to "ON", all the information being saved to PANSAT's storage is compressed.

Inputs - None specifically, this function uses the data obtained from other factions (i.e. send mail or send file).

Processing - If the data compression setting is "ON", all the data sent to the storage system is compressed. The fact that the data is compressed is stored in the information block about the file.

Outputs - The data sent to the file system is compressed.

External Interfaces - All interfacing in this function is done via the SCOS, with the Surrey file operations application, and AX.25 utility packages.

Performance Requirements - The data compression algorithm needs to be optimal towards text, since it is anticipated the majority of files will contain this type of information.

Design Constraints - None.

Attributes - Whether or not to use compression is a stored value set by the BBS control settings block.

Other Requirements - None.

(b) Decompress Data

Introduction - If the mail or file information is flagged that the contents are compressed, the data is decompressed before it is used.

Inputs - The file information and the file contents, if the contents are not compressed, are retrieved from the file system.

Processing - If the data is compressed, as it is read from the file in PANSAT's storage it is run through the decompression algorithm. It is then directed to SCOS, which sends the data to the appropriate connection.

Outputs - The uncompressed mail or file is sent to SCOS.

External Interfaces - All interfacing in this function is done via the SCOS, with the Surrey file operations application, and AX.25 utility packages.

Performance Requirements - The data compression algorithm needs to be optimal towards text, since it is anticipated the majority of files will contain this type of information.

Design Constraints - If the connection is using NPSterm, this function will be skipped, and the compressed file will be sent to the user. This will save effort and time, since the

data would only be recompressed before sending it to an NPSterm connection. Note: the storage's compression routine must be the same as NPSterm's compression routine.

Attributes - None.

Other Requirements - None.

(4) Signal Strength Monitoring

Introduction - The power, or signal strength, of incoming transmissions is checked. If the power is above a threshold, the originator of the signal is warned to turn the transmitting power down. If after three more transmissions the power is still too high, the user is disconnected.

Inputs - For the last received transmission, the signal strength is obtained from the modem and the originators callsign is retrieved from SCOS.

Processing - If the strength checking setting is "ON" and the modem reports a signal strength over the threshold, a counter is set for the connection. A warning is sent to the user to lower transmission power. If after three more transmissions from the user the power is still too high, the user is disconnect (as described in the menu option disconnect section above).

Outputs - If necessary, a warning notifying the user of the too high power transmissions is sent via SCOS.

External Interfaces - All interfacing in this function is done via the SCOS, the modem driver, and AX.25 utility packages.

Performance Requirements - The power level is checked for every connection with every transmission received.

Design Constraints - This task should be concurrent to all other tasks.

Attributes - This function is enabled by the setting in the control block.

Other Requirements - None.

c. Special Notes

(1) File and Mail Numbering. The numbers assigned to files and mail are independent of each other. That is the mail numbered 12 is not the same as a file numbered 12. The program will keep a rotating counter for each, with a range of 0 to 9999. Each new mail or file posted on PANSAT will be assigned the counter value, which will then be incremented by one. Because of deletions, number "holes" will probably appear in directories. For example, mail numbered 12 and 14 may exist, but 13 may have already been deleted. However, due to the autodelete function, by the time the counters toggle, there should be no conflict in numbers.

(2) Fault Tolerance. The spacecraft software will be made using fault tolerant technology, in order to better handle the anomalies of space. The software should be able to detect minor errors, typically radiation-caused "bit flips". The program will attempt to correct the errors and continue normal operations. If the errors are to monumental to continue operations, the ground station should be notified during their next connection of the problems. This will signal that the ground station is to shut down the user services program and reload the software. This is a secondary requirement, and will be implemented after spacecraft's BBS functions are operational. Details for the fault tolerant scheme are specified in Chapter VI, Section B.

2. Ground Station Module

The operator interface for the ground station module will be via a graphical user interface (GUI), centered around multiple windows. Each window in the interface will perform a unique function. Thus, the breakdown of functionality for this module will be centered around the purpose of each specific window, with underlying non-windowed functions listed as they appear. Each one of the windows and the underlying functions will be implemented as concurrent tasks. While the end user of this system is the "ground station" referred to in section B.1.b, to differentiate the actual person using the equipment versus the equipment itself, the individual using the terminal will be referred to as the "operator".

a. Display Terminal Functions

All the functionality for this terminal is incorporated into four windows. When user services is first started up, it launches a monitor program. From this monitor are options to launch any of the four module windows, to quit any of those windows that are executing, to tile or cascade the four windows, or quit the entire program. The monitor will only be able to launch another if that other

window has not yet been run or it has been previously terminated, since only one instance of each window is allowed to be open at a given time. If the monitor program is quit, all of the remaining windows will be terminated. If each window is closed without exiting the program, the monitor program will still remain active until it is explicitly terminated.

(1) Displaying of Telemetry Data

Introduction - The purpose of this window is to graphically display the latest telemetry values obtained from PANSAT.

Inputs - The telemetry values are obtained from the latest telemetry file stored in the archives. A signal is received from the control terminal when an update to the telemetry files is made. The only user input is a "refresh" button on the window.

Processing - When a signal is received that new telemetry values have been saved, the new values are loaded from the disk and converted to a graphical representation of the data. For instance, the battery power will be displayed in a line graph, using a green line for an acceptable power level, a yellow line for borderline power levels, and a red line for unacceptable power levels. At the top of the window, the date and time the telemetry was obtained from PANSAT will be displayed. Once this information has been displayed, no further action will be performed until the next signal is received, indicating a new telemetry has been stored. If the user clicks the refresh button, however, the last telemetry will be redisplayed in the window.

Outputs - A graphical representation of the telemetry is displayed in the telemetry window on the screen.

External Interfaces - The screen, network, and file managers are utilized by this function.

Performance Requirements - The function needs to continually monitor for a "new telemetry" signal on the network. This will allow the window to be updated nearly simultaneous to the data arriving from PANSAT.

Design Constraints - None.

Attributes - The time and date that the telemetry was obtained is stored within the file containing the telemetry information.

Other Requirements - The actual values and format of the telemetry have not yet been defined by the client. Also, the thresholds by which to gage the telemetry values are unknown. Both of these will need to be provided by the client before this function can be implemented.

(2) Tracking of the Satellite. This window is comprised of two displays: a graphic display of PANSAT's position and clock showing the time until the ground station is within the satellite's footprint.

(a) Display The Satellite Track

Introduction - This portion of the display will have a Mercator projection of the world as a background. Plotted on this projection will be the location of the NPS ground station and the current position of PANSAT. Additionally, the track of the satellite will be plotted for its next entire world revolution. The area of the world currently within PANSAT's footprint will be highlighted. The operator may also enter one additional fixed site to plot on the projection as well. The calculated position of PANSAT determined here will be sent to and used by the control terminal and Linux terminal via the network.

Inputs - The Mercator projection is retrieved from a graphic file stored on the hard disk. An initial satellite position to base the position, track and footprint calculation will be obtained from the control panel window on the control terminal. Every time the position is updated in the control panel, a signal is sent notifying this function of the change, at which point the new information is retrieved. The operator also may input the latitude, longitude, and label of an optional fixed site. A refresh button is available to the operator to trigger a redrawing of the entire window.

Processing - When either the window initially starts up or the refresh button in the window is pressed, the window is cleared and the map is displayed. The position of NPS and the optional site, if it was entered in, are plotted on the map.

If the operator changes the values for the optional site, the old site is erased before the new site is plotted. The current position of PANSAT, its track and footprint are calculated and plotted on the map. The position of PANSAT is then sent to the antenna orientation function on the Linux box and the communications module on control terminal. If a signal arrives from the control panel indicating an updated initial satellite position, the new position is retrieved from the control terminal. Once the new position, track and footprint values for PANSAT have been evaluated, the old ones are erased off the map and the new ones are plotted.

Outputs - The positional output is plotted on the map background in the satellite tracking window on the screen. The position of PANSAT is also sent to the control terminal via the network.

External Interfaces - The screen, network, and file managers are utilized by this function.

Performance Requirements - The network is continually being monitored for a new position signal from the control panel. Once it is received, the new satellite position, track and footprint are immediately evaluated. Otherwise, the satellite information is reevaluated and plotted every 20 seconds.

Design Constraints - The color of the footprint should be such that the underlying map will still be visible.

Attributes - The position of the NPS ground station will be hard coded into the software.

Other Requirements - None.

(b) Display Time Until Contact

Introduction - On top of the Mercator projection, the time until the NPS ground station is within PANSAT's footprint is displayed. If an optional fixed site has been entered in, the time until that site is within PANSAT's footprint is also displayed. Both of these values are used by the control terminal as well.

Inputs - The current satellite and optional fixed site positions are obtained from the function listed in the previous section.

Processing - Based on the positional data, the times until the ground station and optional site, if available, are within PANSAT's footprint are calculated and sent to the display. When these two times are calculated, they are sent to the communications module on the control terminal and the antenna orientation function on the Linux terminal.

Outputs - The time is sent to the satellite tracking window on the screen and the control terminal via the network.

External Interfaces - The screen and network management utilities are used by this function.

Performance Requirements - Once per second, the time on the display is updated, which makes it appear as a continual countdown.

Design Constraints - Since a new satellite position is calculated only once per 20 seconds, the times until the ground station and the optional site are within the footprint should also be calculated once per 20 seconds. These calculated values are the values sent to the control terminal. Until a new position is determined, however, the time on the display is simply counted down.

Attributes - None.

Other Requirements - None.

(3) Communications Repeater

Introduction - This window merely acts as a repeater or "mirror" of the communications to and from PANSAT. The default is to show only the communications between the station operating this software module, such as the ground station, and the satellite. However, the operator can specifically list the callsign or callsigns whose communications are to be displayed, or the operator can choose to display all the communications to and from PANSAT.

Inputs - The callsign of the station operating the software is obtained from the control terminal. All the packets going to and from PANSAT are provided from the SCC (the modem controller) utilities. The list of callsigns whose communications are to be displayed is obtained via user input.

Processing - When a packet of data is received, it is immediately checked with the filter settings. If the callsign of the originator, for a packet to PANSAT, or the callsign of the recipient, for a packet from PANSAT, matches a callsign set by the operator, the packet is copied to the display. Otherwise, the packet is ignored. This process is a continual loop.

Outputs - The communication that meets the filter criteria is sent to the communication repeater window. The latest data is printed at the bottom of the window, scrolling the older information up, and eventually off, the window.

External Interfaces - The network, screen, SCC/modem and AX.25 manager utilities are utilized by this function.

Performance Requirements - The communications to or from PANSAT should be sent to the display almost immediately after it is received by the modem, if the communication's callsign qualifies to be displayed per the operator set filter. Thus every packet received should be processed immediately by this function. This equates to less than a one second turn around from receipt to display.

Design Constraints - The default callsign to pass through the filter, or to be sent to the display, is the station's callsign which is operating this software. This default is set when the window is launched.

Attributes - None.

Other Requirements - None.

(4) Archive Management Functions. The archive files are hierarchally organized by date (year, month, and day subdirectories). In each day's subdirectory, all the mail, files telemetry, and mail/file listings that were downloaded during the day are stored. Telemetry and listings are always received, which is set up in a default batch command, but the other elements are obtained only when requested by the operator. The file comprising the archives are stored on the server computer. This window allows five basic operations on the archive files.

(a) Display A Day's Directory

Introduction - This function lists out all the mail, files, telemetry, and mail/file listings obtained from PANSAT on the day specified by the operator. The operator may select, or choose, one of these entries. The other functions in this section will use this selection.

Inputs - First, the day to list is inputted by the operator. The list of entries for this day is obtained from the file system. The user can then pick the entry to select.

Processing - Using the day supplied by the operator, the day's list is obtained from the file system. It is sent to the screen, with no entry being highlighted. The operator can then select an entry. The entry selected is then updated in the list - it is now shown highlighted. If user chooses the entry again, it is unselected and the highlighting is removed from it. The user can also choose another entry, which removes the selection and highlight from the first entry, then selects and highlights the new entry.

Outputs - The list of entries is sent to the archive window on the screen. The entry the user selects is specially marked.

External Interfaces - The screen and file system management utilities are utilized by this function.

Performance Requirements - None.

Design Constraints - Highlighting should make the entry boldface in the listing. Since the search function can also select an entry, only one of the search or directory functions can be viewed at one time.

Attributes - None.

Other Requirements - None.

(b) View File Contents

Introduction - The contents of the entry that was selected in either the directory or search functions is displayed. If the contents are too large to fit in the window at one time, the user can use scroll bars to view up and down the contents.

Inputs - The desired file is read from the file system.

Processing - The file read is merely redirected to the window on the screen. The scroll bars feature is inherent in the Windows 95/NT architecture.

Outputs - The file is sent to the archive window on the screen.

External Interfaces - The screen and file system management utilities are utilized by this function.

Performance Requirements - None.

Design Constraints - Once a new file is selected by the operator, the old file is erased from the window. But as long as the same entry remains selected, it will continue to be displayed in the window.

Attributes - None.

Other Requirements - None.

(c) Print File Contents

Introduction - The file that was selected by the directory or search functions is sent to the printer to make a hard copy.

Inputs - The selected file is read in from the file system.

Processing - The file inputted is merely directed to the print manager for output.

Outputs - The entire file is sent to the print manager.

External Interfaces - The file system and print manager utilities are utilized by this function.

Performance Requirements - None.

Design Constraints - The print manager handles all the hardware particulars of the printer. The data must be in the desired format when sent to the print manager, however.

Attributes - None.

Other Requirements - None.

(d) Delete a File

Introduction - The operator of the terminal may delete a file that has been selected by the directory or search functions. Before the deletion occurs, the operator will be prompted to ensure that the file is meant to be deleted. This feature would be typically be used only if the contents of a file were garbled during transmission and contained no useful information.

Inputs - The answer to confirm the deletion is received from the user.

Processing - If the answer to the question is affirmative, the file is deleted from the system. This list of entries on the screen is updated and will not include the file that was just deleted. Also, since the selected file was just deleted, no entry will be selected.

Outputs - The query confirming the delete is sent to the dialog box on the screen. The command to delete the file is sent to the file system. The list of entries on the screen is updated on the archive window.

External Interfaces - The file system and screen management utilities are utilized by this function.

Performance Requirements - None.

Design Constraints - The deleted file should be put into the Windows 95/NT recycle bin. That way it can be undeleted if need be.

Attributes - None.

Other Requirements - None.

(e) Search Though Archives

Introduction - Instead of listing out the entries for a particular day, as performed above, the operator can list out files that meet a certain criteria, even if the files come from different days. The operator can then select one entry from this list, just as in the directory function. The criteria to list the entries are: the beginning and ending dates to search within, pattern matching within the file, type of file (i.e. telemetry, file listings or mail), or any combination thereof.

Inputs - First, the search criteria is entered in by the operator. The list of entries matching the criteria is obtained from the file system. The user can select one of the entries.

Processing - After the criteria is entered by the operator, the directory for every day in the range specified is examined. If no range is specified, the entire archive is searched. Within each day's directory, only the files with matching types to the selection are examined. If no type is

specified, all the files are examined. Within each file examined, the pattern is searched for. If the pattern is found, the file is added to the list to select from. If no pattern is specified, the file will automatically be included in the list. Once all the files have been examined, the list of matches is sent to the archive window. The operator may then select one of the entries, which highlights the file in the list. If the operator selects the same entry, it is unselected and the highlighting is removed. If the operator selects another file, the first file is unselected and the highlighting is removed, then the second file is selected and highlighted.

Outputs - The list of entries matching the search criteria are displayed in the archive window on the screen. If an entry is selected, the list is updated with the selected entry being highlighted.

External Interfaces - The screen and file system management utilities are utilized by this function.

Performance Requirements - The search results should be displayed as quick as possible, even for the pattern matching criteria.

Design Constraints - Highlighting should make the entry boldface in the listing. Since the directory function can also select an entry, only one of the search or directory functions can be viewed at one time.

Attributes - None.

Other Requirements - None.

b. Control Terminal Functions

The control terminal executes several windows, as well as a few underlying functions. These underlying operations need to continue working, even if all the windows are terminated. To support this infrastructure, when the user services program first begins, it launches a monitor program. From the monitor, the operator can open any of the windows, which are described below. This monitor will allow the user to open a window that was previously terminated, since only one instance of each window is allowed to be open at a given time. If all the windows are shut down, the monitor remains active along with the underlying functions. The monitor will also allow the operator to execute other applications on top of the user services program. However, these applications will be independent from the monitor. Once the application is launched, the user services program will not be "aware" of it. This is essentially a "launch and forget" policy. The user services program will only be terminated when the monitor is shut down. When exiting the monitor, all of the windows remaining open will be terminated, the underlying functions will cease and the program will terminate.

(1) Communications Window. This window is the centerpiece of the control terminal. This is the window by which the operator of the terminal directly interfaces with PANSAT. All the sub functions for the communications window are mutually exclusive, with the exception of write mail and read stored mail functions (see corresponding sections below). That is, to avoid race condition errors while interacting with PANSAT, a communications window function may operate only after the other communications window functions have completed.

(a) Interaction Interface

Introduction - This is the central feature of the communications window. This displays the communications to and from PANSAT in an easy-to-read, scrolling format. All the possible commands to send to PANSAT are displayed as option buttons. The operator simply presses the desired command button and a dialog box will request any additional information required by the command (i.e. a list files command would then request the range of files to list). After the command has been accepted, it is sent to the satellite and displayed as outgoing communications in the window. The commands options possible are listed in Table 1. If the terminal is in super user mode, the commands in the Table 2 are also listed. Any additional actions or information not explicitly listed in the either Table 1 or 2 is described in the remainder of this section.

Inputs - The commands to send to PANSAT are obtained via user input. The communications from the satellite are obtained via the SCC and modem.

Processing - Everything received from the satellite destined for the callsign of the station operating this software is sent to the display and processed accordingly. Once the complete command has been entered by the operator, it is processed then sent to the display and PANSAT simultaneously. All the processing is described in the remaining subsections the communications window. If not

listed there, no actual processing of the data is done, it is only sent to the display and, if appropriate, PANSAT.

Outputs - All input, be it from the operator or PANSAT, is sent to the communications window on the screen. Additionally, if the input is a command from the operator, it is sent to PANSAT via the modem.

External Interfaces - The screen, SCC/modem and AX.25 manager utilities are utilized by this function.

Performance Requirements - Due to the limited window of opportunity to communicate with PANSAT, the entering in of commands, the processing of the data, and the displaying of communications should take as little time as possible.

Design Constraints - The communications window should look and act the same whether an ASCII interface protocol or NPSterm is used. The only difference is that using the NPSterm protocol will provide faster communications and some additional flexibility.

Attributes - The packets which are destined for the station operating this software is determined by the callsign, which is set in this window (see subsection below).

Other Requirements - The connect command is always available, unless the ground station is already connected

with PANSAT. This, along with write mail and read stored mail, are the only options available when not connected to the satellite. Once a connection has been established, however, all the commands except for connect become active until the disconnection. The connect command will remain available if no connection is ongoing, even if the tracking data indicates the ground station is not in the satellite's footprint. This allows the operator to still be able to connect with PANSAT, even if the tracking data is in error.

(b) Update BBS Control Settings Onboard PANSAT

Introduction - See Table 2. When the operator presses the update control settings button, all the values from the control panel window are obtained, then sent to PANSAT.

Inputs - The user enters the update command. The values are obtained from the control panel function.

Processing - Once the button is pressed the settings are obtained, concatenated together, and duplicated. The settings are then relayed to PANSAT. Control is then passed to the verification function, which will properly respond to PANSAT's query. After verification, this function ends.

Outputs - The command with all the settings, which are in duplicate, are sent to PANSAT via the modem.

External Interfaces - The screen, SCC/modem and AX.25 manager utilities are utilized by this function.

Performance Requirements - None.

Design Constraints - The settings are sent in duplicate to ensure proper receipt of the data by PANSAT.

Attributes - None.

Other Requirements - The terminal must be in super user mode to select this command, otherwise the button would not be available. Furthermore, the verification function necessary to confirm this command would only be operational in super user mode.

(c) Getting BBS Control Settings From PANSAT

Introduction - See Table 2. This command is sent up to PANSAT. The satellite sends back the current values it has for the BBS control settings. These settings are displayed in a new window, with the values that are different from the control panel settings highlighted in yellow. The operator is then prompted to replace the settings in the control panel window with the settings from PANSAT. If affirmative, the settings in the control panel are overwritten. Otherwise, the values are ignored.

Inputs - The command and "overwrite control panel" answer are entered in by the user. The settings are obtained from PANSAT via the modem.

Processing - Once the command button has been pressed, the command is sent to PANSAT. The verification function is called to respond to PANSAT's query. Once control is returned and the settings have been received, the duplicate values are checked. If the values are the same, they are displayed in a new window. As they are being displayed, they are compared with the values in the control panel. If these values differ, the new value is displayed in yellow. If the operator chooses to update the control panel, then the values are sent to the control panel. Otherwise, the values are ignored and disappear from the system as soon as the new window is closed. If the duplicated values are different, however, the operator is warned of the error, then the information is ignored

Outputs - The command is sent to PANSAT via the modem. The operator is asked the "overwrite control panel values" question or is told of the value errors via a dialog box on the display.

External Interfaces - The screen, SCC/modem and AX.25 manager utilities are utilized by this function.

Performance Requirements - None.

Design Constraints - The duplicate values are used to ensure proper receipt of the control settings from PANSAT.

Attributes - None.

Other Requirements - The terminal must be in super user mode to select this command, otherwise the button would not be available.

(d) Write Mail

Introduction - This option may be selected at any time, whether connected to PANSAT or not. When the operator chooses this option, a dialog box comes up. The user fills in the "to", "subject" (which is optional), and "message" fields. Once the mail is complete, the operator presses the done button and the message is stored until the send mail button is pressed, which is only available when the terminal is connected to PANSAT. Only at that time is the mail actually sent to the satellite.

Inputs - The command and mail message are entered in by the user.

Processing - Once the command is received, the dialog box is sent to the display for the user to fill out. Once the operator is finished entering the message, which is indicated by the done button of the dialog box being pressed, the mail is put in the mail format described in Table 1. Lastly, the

mail message is appended at the end of the file which contains the mail message buffer.

Outputs - The mail message is appended to the end file containing the mail message buffer. The mail message is in the Table 1 format, including the send mail command line.

External Interfaces - The screen and file system manager utilities are utilized by this function.

Performance Requirements - This is the one of the two functions in the communications window section that can operate concurrently with the other functions. All other functions, except for read stored mail, are mutually exclusive.

Design Constraints - The mail buffer should be implemented as a file on the hard disk. Using this, no practical limit to the number of mail messages that could be stored for transmission would exist. Furthermore, the buffer would operate as a first-in-first-out queue. The first mail message written would be the first transmitted to PANSAT; the last message written would be the last sent.

Attributes - The filename to store the mail messages would be a hard coded program constant. All the mails are stored in a single file in proper format to send to PANSAT. Then when the send mail button is pressed, all the mail messages are streamed to the spacecraft.

Other Requirements - The only time a mail message could not be written is when the buffer is actively being sent to PANSAT. The best time to write a mail message, however, is when the ground station is not within the satellite's footprint. This would allow for the best possible utilization of the limited communication window.

(e) Send Outgoing Mail

Introduction - When the send mail command is chosen by the operator, the mail message buffer is sent to PANSAT, each message being in the format listed in Table 1. This command will not allow the operator to write the mail, that is accomplished in the previous section.

Inputs - The command is entered in by the operator. The mail message buffer is read from a file on disk.

Processing - Once the command is received, the mail message buffer is sent to PANSAT, one message at a time. Although to the operator, it appears as if all mails are being sent together since no interaction is allowed until all the messages are sent. If the buffer is empty, the operator is warned that no messages were queued. Once all the mail in the buffer has been sent to PANSAT, the buffer is erased.

Outputs - The contents of the mail message buffer file is sent to PANSAT via the modem. If no mail messages are

queued, a warning is sent to the user in a dialog box on the display.

External Interfaces - The screen, file system, SCC/modem and AX.25 manager utilities are utilized by this function.

Performance Requirements - None.

Design Constraints - The mail messages are sent one at a time to allow for proper parsing by PANSAT. PANSAT will not be able to tell the difference if the mail messages had been typed in by a user at that time or earlier

Attributes - The filename where the mail messages are stored will be a program constant.

Other Requirements - None.

(f) Store Incoming Mail

Introduction - When a mail message is received from PANSAT, it is scrolled across the communications window just like most other communications from the satellite. The operator does not need to take the time to read the messages at that time, however. While the messages are sent to the display, they are simultaneously being saved in a "mail box". The operator can read the "mail box" using the function described in the next section.

Inputs - The mail message is received from PANSAT via the modem.

Processing - The data obtained while receiving a mail message is appended to the file containing the "mail box".

Outputs - The mail message is appended to the end of the "mail box" file, via the file system.

External Interfaces - The screen, file system, SCC/modem and AX.25 manager utilities are utilized by this function.

Performance Requirements - None.

Design Constraints - If the "mail box" is currently being read, the mail will be stored in a temporary file until the "mail box" file is released from the reading function. At this time, the temporary file will be concatenated to the end of the "mail box" file.

Attributes - The filenames for the "mail box" and temporary files will be hard coded program constants.

Other Requirements - None.

(g) Read Stored Mail

Introduction - When this operation is selected, a dialog box opens displaying the first mail stored in the "mail box".

The operator can then save the mail in a file on disk, print the mail message, display the next mail message in the "mail box" or exit the read stored mail function. When "the display next mail massage" option is selected, the old mail is deleted from the "mail box".

Inputs - Each mail is read from the "mail box" file via the file system. The operator then enters their choice from the operations listed.

Processing - The first mail in the "mail box" is read and displayed for the operator. If the operator chooses to save the mail in a file, the name of the file is requested. If the print option is chosen, the mail message is sent to the print manager. If the "display the next mail" option is chosen, the next mail in the "mail box" replaces the one displayed in the dialog box. Additionally, the old mail is removed from the front of the "mail box" file. If the "display the next mail" option is selected and there are no more mail messages in the "mail box", the dialog box closes, the "mail box" is emptied, and a new dialog box notifies the operator that no mail messages are in the "mail box". This last dialog box is also displayed if the "mail box" is empty when the read stored mail command is first selected. If the operator exits the window, the mail message currently on the display is not deleted from the "mail box".

Outputs - Each mail is sent to a dialog box on the display.

External Interfaces - The screen, print and file manager utilities are utilized by this function.

Performance Requirements - None.

Design Constraints - This function will not be selectable if the save incoming mail function (see previous section) is actively adding mail to the "mail box."

Attributes - The filename for the "mail box" will be a hard coded program constants.

Other Requirements - None.

(h) Upload A File

Introduction - When this command is entered, a dialog box requests the extra information specified in Table 1. After that information is entered, a second dialog box pops up requesting the filename as it is appears on the local hard drive. When the name is entered, the file transfer begins. While the transfer is ongoing, a dialog box showing the progress of the file transfer in a line graph is displayed.

Inputs - The operator enters in the filename. The file to be transferred is obtained via the file system.

Processing - After the command is entered by the operator, the local name of the file is requested. The name of the file

as it will be stored on PANSAT is the same as it is on the local drive. After it is determined the file to send exists locally, the command is sent to PANSAT. The file is then opened and the contents are sequentially sent to PANSAT, finishing with the end-of-file character. At the beginning of the file transfer, the file size is retrieved from the file system. After each packet is sent to PANSAT, the cumulative amount of data sent is divided by the file size, resulting in the percentage of the file that has been transferred. This percentage is displayed for the operator.

Outputs - The file is sent to PANSAT via the modem. The dialog boxes are sent to the display.

External Interfaces - The screen, file system, SCC/modem and AX.25 manager utilities are utilized by this function.

Performance Requirements - The percentage of the file transfer completed should be updated after the transmission of every packet of data.

Design Constraints - A file transfer protocol similar to the PACSAT level 0 ftp will be implemented for PANSAT.

Attributes - None.

Other Requirements - None.

(i) Download A File

Introduction - After the normal dialog box obtains the parameters for the command, an additional dialog box requests the operator to provide the local name for the file. The file is then downloaded from PANSAT and saved in a file on the hard disk.

Inputs - The file is received from PANSAT via the modem.

Processing - After the operator provides the local name for the file, the file is created. If the file already exists, the operator chooses to either overwrite the existing file, append to the end of the existing file, or specify a new local filename. Only then is the command sent to PANSAT. PANSAT will reply with the number of bytes in the file, then send the file contents. All the data from PANSAT after this point, and until number of bytes specified is received, is considered part of the file and is directed into the open file. Once the transfer is complete, the file is closed. While the transfer is ongoing, a flashing box on the display indicates that file downloading is taking place.

Outputs - The data received from PANSAT is stored in the open file via the file system. The flashing box is sent to the display.

External Interfaces - The screen, file system, SCC/modem and AX.25 manager utilities are utilized by this function.

Performance Requirements - None.

Design Constraints - A file transfer protocol similar to the PACSAT level 0 ftp will be implemented for PANSAT.

Attributes - None.

Other Requirements - None.

(j) Post the Broadcast Message

Introduction - After this command is selected, a dialog box will request the name of the file containing the broadcast message. The broadcast message needs to have been previously saved in text format, such as if it were written in the Windows 95/NT Notepad utility, on the hard drive. After the filename is entered in, the message will be updated onboard the satellite.

Inputs - The operator enters the filename of the broadcast message. The broadcast message is then obtained via the file system.

Processing - After the command and filename are entered in, the command is sent to PANSAT. The verification function is then called to respond to the satellite's query. After the verification has been sent, then the file is sent to PANSAT.

Outputs - The file is sent to PANSAT via the modem.

External Interfaces - The screen, file system, SCC/modem and AX.25 manager utilities are utilized by this function.

Performance Requirements - None.

Design Constraints - By choosing the filename, the operator can store several broadcast messages and select the appropriate one. Additionally, two or more messages can be concatenated into one file, then be sent as a single message to PANSAT. This would necessary if a new message wanted to be added to the end of the old message, since PANSAT simply replaces the old broadcast message with the new one.

Attributes - None.

Other Requirements - The terminal must be in super user mode to select this command, otherwise the button would not be available.

(k) Process A "Delete Mail/File" Command

Introduction - This command is handled differently depending on the mode the terminal is in. If the terminal is in normal mode, the delete command will behave exactly as it would for a general user accessing PANSAT (see Table 1). If the terminal is in super user mode however, the

command will unconditionally delete every file specified by the ground station (see Table 2). In both cases, the system merely asks for the parameters of the delete command. However, in super user mode a notice is displayed in the dialog box reminding the operator of the special delete ability.

Inputs - The parameters are entered in by the operator.

Processing - After the information is obtained from the user, the command is sent to PANSAT. However if in super user mode, the text "SU" is added to the end of the command line. The verification function is then called. This "SU" flag notifies PANSAT of the unconditional delete option for this command.

Outputs - The command is sent to PANSAT via the modem.

External Interfaces - The screen, SCC/modem and AX.25 manager utilities are utilized by this function.

Performance Requirements - None.

Design Constraints - This dual functionality for the delete command allows the operator to process mail, specifically this is the removing of the ground station's callsign from the "still to" list onboard the satellite, without unintentionally purging the mail from PANSAT.

Attributes - None.

Other Requirements - None.

(l) Process A "Terminate User Services" Command

Introduction - See Table 2. This function would typically be used only when the ground station wants to replace the spacecraft's user services program with a new version. After this command is entered, the operator is queried to double check their intentions. If the double check is affirmative, the user services program aboard PANSAT is shut down.

Inputs - The operator enters their answer to the double check question.

Processing - When the dialog box is sent to the operator to verify operator intentions, the terminal will also generate several beeps. This should attract the attention of anybody else in the office, ensuring that the operator is double checked by other people. After an affirmative response to the double check question has been received from the operator, the command is sent to PANSAT and the verification function is called. After the satellite has been sent verification answer, the function ends.

Outputs - The double check query is sent to the display. The command is sent to PANSAT via the modem.

External Interfaces - The screen, SCC/modem and AX.25 manager utilities are utilized by this function.

Performance Requirements - The "bells and whistles" should go off for a few seconds before the operator is allowed to answer the double check question. This not only will get the attention of the entire office, but should reinforce the seriousness of the proposed course of action.

Design Constraints - After this command is executed, the spacecraft user services software must be reloaded to become operational again. The ground station will have a special program that loads executables to the satellite. This program is separate from the user services software, but may be executed via the application launcher function.

Attributes - None.

Other Requirements - The terminal must be in super user mode to select this command, otherwise the button would not be available.

(m) Execute Batch Commands

Introduction - If a batch job is enabled (see the batch job editor window below), the batch job is executed as soon as the NPS ground station is first within PANSAT's footprint. When the batch job begins to execute, the operator will no longer be able to enter any of the commands in the

communication window, with the exception of write mail and read stored mail. Once the batch job completes, control is returned to the operator and the commands will be enabled.

Inputs - The time that the NPS ground station is within the satellite's footprint is obtained from the display terminal, via the network. The batch job commands are read from a file tagged by a batch job flag, via the file system.

Processing - When the time switches from "time until within PANSAT's footprint" to "within PANSAT's footprint", the batch job flag is checked. If it is enabled, the commands the communications window are suspended. The file pointed to by the batch job flag is opened and the commands are read in. The commands are simultaneously sent to the communication window on screen and to PANSAT. After a command has been sent to PANSAT, the batch job pauses until the satellite is ready for the next command. The next command in the file is then read and the process repeats. Once the entire file is read, the file is closed and the command buttons are enabled once again.

Outputs - The commands read from the file are simultaneously sent to the communications window on the display and to PANSAT via the modem.

External Interfaces - The screen, file system, SCC/modem and AX.25 manager utilities are utilized by this function.

Performance Requirements - The commands cannot be "shotgunned" at the satellite. The system must wait for a response from PANSAT before the next command can be sent.

Design Constraints - When the command prompt is received from PANSAT, this batch command processor knows the satellite is ready for the next command.

Attributes - The batch job flag is set in the batch job editor.

Other Requirements - Before it can be executed, the batch file must be "compiled" or checked to make sure that all the commands are complete and make sense (i.e. no commands should follow disconnect). This "compiled" flag is skipped over when reading the file.

(n) Set The Ground Station's Callsign

Introduction - This allows the operator to change the callsign identity of the ground station. It will be updated onboard PANSAT as well as locally. The operator enters in the new callsign via a dialog box, then is prompted to make sure the new callsign is correct. The new callsign will take effect as soon as the ground station is disconnected from PANSAT.

Inputs - The user enters the new callsign. The response from PANSAT is received via the modem.

Processing - After the operator enters the new callsign, a dialog box displays the new callsign and asks the operator if the new callsign is correct. If it is, the new callsign is sent in triplicate to PANSAT. If PANSAT responds that the new callsign has been accepted, then the new callsign is set to take effect as soon as the operator disconnects. If PANSAT does not accept the callsign, nothing will be updated locally. If PANSAT reports that the callsign has been changed, but to not to the callsign specified locally, the local callsign will be changed to the wrong callsign until the operator can update PANSAT with the correct callsign. Once the new callsign takes effect, it is sent to the display terminal in order to update the communications repeater window.

Outputs - The new callsign is sent to PANSAT via the modem. It is sent to the display terminal via the network.

External Interfaces - The screen, network, SCC/modem and AX.25 manager utilities are utilized by this function.

Performance Requirements - None.

Design Constraints - The callsign is sent in triplicate to ensure its proper receipt by PANSAT. If a wrong callsign is assumed by PANSAT, the ground station will assume that wrong callsign so that PANSAT will still be able to recognize the ground station. If the ground station does not assume the wrong callsign, PANSAT will never recognize

the ground station as such until the spacecraft's user services program is reset.

Attributes - The initial ground station callsign will be "KD6CXV".

Other Requirements - The terminal must be in super user mode to select this command, otherwise the button would not be available.

(o) Verification Function

Introduction - After a command from Table 2 is sent to PANSAT, the satellite first ensures the command was received from the ground station's callsign. Next, to ensure the validity of the ground station, PANSAT sends it a series of random numbers. The next data sent up from the ground station must be the answer to these numbers after applied through a set of functions. If the wrong answer is received, the spacecraft ignores the command previously sent up and logs the incident. This function is automated - the operator is not involved with the response. It receives the numbers from the satellite and sends back the correct reply.

Inputs - The series of random numbers is received from PANSAT via the modem.

Processing - This function only operates if the terminal is in super user mode. The numbers received from PANSAT are

in triplicate. If there was an error in the transmission, the best-two-out-of-three rule will be used to determine the numbers. Once the numbers are determined, the response is immediately sent back in triplicate to PANSAT. After that, program control is returned to the function that called this verification procedure.

Outputs - The answer in triplicate is sent to PANSAT via the modem.

External Interfaces - The SCC/modem and AX.25 manager utilities are utilized by this function.

Performance Requirements - None.

Design Constraints - To make the verification function less cumbersome for the satellite's process, the actual function will use only integers and bit level operations.

Attributes - For security, the actual function will not be published, but will be handed over to ground station personnel at the time of software turnover.

Other Requirements - The verification query and answer are sent in triplicate to ensure proper reception of each.

(p) NPSterm Version Check

Introduction - Once the command for the communications between PANSAT and the ground station to use NPSterm has been sent to the satellite, it returns the NPSterm version number used by the spacecraft user services program. If the version is different from the version being used by the ground station, the NPSterm session is aborted. The operator is warned about the discrepancy. Communications with PANSAT may continue, but only using the ASCII interface.

Inputs - The NPSterm version number is received from PANSAT via the modem.

Processing - The version number received from PANSAT is compared to an internal NPSterm version number. If they are the same, the connection will continue operations using the NPSterm protocol. If the numbers are different, however, the "switch to ASCII" command is sent to the spacecraft. A dialog box is sent to the display warning the operator that the NPSterm versions differ and that the mode is switching to the default ASCII protocol for communications with PANSAT.

Outputs - If the versions differ, the warning dialog box is sent to the display. The "switch to ASCII" command is sent to PANSAT via the modem.

External Interfaces - The screen, SCC/modem and AX.25 manager utilities are utilized by this function.

Performance Requirements - None.

Design Constraints - In order to gracefully exit NPSterm, the "switch to ASCII" command must be always be accepted by PANSAT, no matter what terminal mode the satellite has set the connection to be in.

Attributes - The NPSterm version number of the ground station will be a hard coded program constant.

Other Requirements - None.

(q) Display Time Until Satellite Contact

Introduction - On top of the communications window, the time until the NPS ground station is within PANSAT's footprint is displayed.

Inputs - The time to display is received from the satellite tracking window on the display terminal via the network.

Processing - The time is received from the display terminal every twenty seconds. When it is received, the clock time is updated. For the time in between the receipts, the clock time is merely counted down.

Outputs - The time is sent to the communications window on the screen.

External Interfaces - The screen and network manager utilities are utilized by this function.

Performance Requirements - Once per second, the time on the display is updated, which makes it appear as a continual countdown.

Design Constraints - None.

Attributes - None.

Other Requirements - None.

(b) Control Panel

Introduction - This window displays the BBS control settings for PANSAT as listed in Table 3. The operator can modify the values in this window. When the "update BBS control settings" command is selected in the communications window, the values displayed in this window are the values sent to PANSAT. If the settings are obtained from PANSAT, the operator has the choice of overwriting the values currently in the control panel with the new ones.

Inputs - Modifications to the control settings are entered by the operator or by the new settings window from the "get BBS control settings" operation.

Processing - The operator can update the settings at any time, as long as the terminal is in super user mode. When the control settings are updated by the "get BBS settings" operation, the control panel is redisplayed with the new values. Whenever the satellite position fields are updated, the information is sent to the satellite tracking window on the display terminal. If the satellite position data is older than four weeks (28 days), the fields will be highlighted in red to indicate that the data needs updating.

Outputs - The satellite position fields are sent to the display terminal via the network.

External Interfaces - The screen manager utilities are utilized by this function.

Performance Requirements - The current date and time fields displayed in the control panel should be updated every second with the values from the system clock.

Design Constraints - While the control panel can be viewed in any mode, the terminal must be in super user mode to modify the control settings.

Attributes - The default values for the settings are as listed in Table 3.

Other Requirements - None.

(3) Batch Job Editor. This window has a vertically split screen. On the left side, which will encompasses about 20% of the window, the list of all the batch jobs available is always displayed. The right side is used to view and edit the contents of batch files. Additionally, at the top of the window is a status line indicated whether a batch job is enabled or not.

(a) Select A Batch Job

Introduction - The operator may select or choose one of the batch jobs listed on the left part of the split window. Only one job can be selected at a time. This selection is then used by the communications window.

Inputs - The operator indicates the job to select.

Processing - Once a batch job has been selected by the user, it will be highlighted on the display. If the operator selects the entry again, it will unselect the batch job and the highlighting will be removed. If the operator selects another entry while one is already selected, the original selected batch job will be unselected and unhighlighted, and the new entry will be selected and highlighted.

Outputs - The batch job that is selected is updated in the batch job window by being highlighted.

External Interfaces - The screen manager utilities are utilized by this function.

Performance Requirements - None.

Design Constraints - Highlighting should make the entry boldface in the listing.

Attributes - None.

Other Requirements - None.

(b) Display A Batch Job

Introduction - The contents of the selected batch job file are displayed in the right portion of the window. If the contents are too large to fit in the window at one time, the operator can use scroll bars to view up and down the contents.

Inputs - The command choice is entered by the operator. The file containing the batch job is read via the file system.

Processing - The file indicated by the selection choice is opened and displayed to the window on the screen. The scroll bars feature is inherent in the Windows 95/NT architecture.

Outputs - The batch job file is sent to the right side of the batch job window.

External Interfaces - The screen and file system manager utilities are utilized by this function.

Performance Requirements - None.

Design Constraints - Once a new batch job is selected by the operator, the old job that is being displayed is erased from the window.

Attributes - None.

Other Requirements - None.

(c) Enable/Disable A Job

Introduction - If the operator chooses this function, the selected batch job will be enabled. Once a batch job has been enabled, it will be executed the next time the ground station is within PANSAT's footprint (performed by the communication window).

Inputs - The operator enters the command.

Processing - The batch job flag is set to indicate the selected entry. If no job is selected, a warning is sent to the operator. A special symbol is put next to the enabled job. The status line at the top of the window is updated to "ENABLED". If the user chooses this function and the selected job is already enabled, that entry will then be

disabled - the special symbol will be removed, the batch job flag will be erased, and the status line will be updated to "DISABLED". If an entry is enabled while another job is already enabled, the new file will be enabled and the old file will be disabled. The special symbol will be removed from the old entry and placed next to the new one. The status line will not need to be updated, but the batch job flag will need updating.

Outputs - The special symbol and the status line are sent to the batch job window on the display.

External Interfaces - The screen manager utilities are utilized by this function.

Performance Requirements - The enabled batch job will take affect the next time the ground station enters PANSAT's footprint.

Design Constraints - Only one batch job can be enabled at a time. If the batch job does not have the special "compiled" flag at the beginning of the file, it cannot be enabled and a warning will be sent to the operator.

Attributes - The special symbol placed next to the enabled batch job should an easy to see arrow which would leave no question as to which file has been enabled.

Other Requirements - None.

(d) Edit A Job

Introduction - If a batch job is selected when this function is chosen, that file is displayed on the right section of the batch job window and the cursor is placed at the beginning of the file. The operator then can edit the file much like using the Windows 95/NT Notepad program. The operator can also insert automatically formatted commands at the point of the cursor. When the editing is complete, a done button is pressed and the batch job is saved. If no batch job was selected when this function was chosen, the operator is queried if the system should create a new batch job. If affirmative, the batch job name is inputted by the operator, then a blank batch job is opened, ready to edit.

Inputs - If the batch job exists, it is loaded from the file system. Otherwise the new batch job name is entered by the operator. The batch job is edited by the operator.

Processing - Once a new batch file name is provided, it is added to the batch job list on the left side of the batch job window. When an existing batch job is opened for editing, the special "compiled" flag at the beginning of the file, if it exists, is removed. Editing any batch job file will be in a typical text editor style. A special buttons will be at the top of the editing box: "INSERT COMMAND". The insert command button brings up the list of the Table 1 commands. The operator can choose one of the commands. Corresponding to the command, a dialog box requests any

extra information that the command requires (i.e. the range of mails to list for the list mail command). Once the information has been entered, the dialog box disappears and the command, in perfect format, is inserted at the point of the cursor's position. The cursor is then moved to the next line.

Outputs - If the batch file exists, it is sent to the editing box, which is the right section of the batch job window. After editing, the batch job will be saved to a file on disk via the file system.

External Interfaces - The screen and file system manager utilities are utilized by this function.

Performance Requirements - None.

Design Constraints - A single line may only contain one command. The next command must be on the next line.

Attributes - Comments begin with a "//" and continue to the end of the line. A comment could be on its own line or at the end of a line with a command in it.

Other Requirements - Only Table 1 commands can be entered since only those commands can be executed in a batch file. Super user commands (Table 2) must be sent manually to PANSAT by an operator.

(e) Compile A Job

Introduction - Once a batch job has been edited, before it can be enabled, it must be "compiled". Actually, this process only checks the format and syntax of the file to ensure that it can be executed without errors. The selected batch job will be checked. With the first error found, "compilation" will cease and the batch job being checked will be put into the edit mode. In the editing portion of the batch job window, the cursor will be placed on the line with the error. Additionally, the window will be displayed with a description of the error. As soon as the operator moves the cursor, this error message will be replaced with the status message that was overwritten. When a successful compilation occurs, a dialog box announces the fact.

Inputs - The batch job selected is retrieved from the file system.

Processing - If no batch job is selected, the operator is warned that an entry must be selected to "compile". Otherwise, the file containing the selected batch job will be opened. Each line of the file will be checked to make sure it is in the format listed in Table 1. Any command not in Table 1 or any variation from the syntax listed therein will cause an error. The only exception to this is the read and send file commands, which will have a local filename appended to the end of the line. If "C PANSAT" is not the first command in the file, an error will be raised. Any

command after a disconnect command will cause an error. If an error is raised, the file is sent to the batch job editing function, the cursor is placed on the line causing the error, and the error message is displayed. After the file is fixed, it must be "compiled" all over again. If the compilation was successful, the operator is notified of this and a "compiled" flag is inserted at the beginning of the file containing the batch job. If a "compiled" flag is already at the beginning of the file when this option is chosen, it is removed from the file and the process continues.

Outputs - If an error occurs, a descriptive error message is sent to a dialog window. Otherwise, when compilation is successful, a dialog box is sent to the display and the "compiled" flag is inserted at the beginning of the file containing the batch job.

External Interfaces - The screen and file system manager utilities are utilized by this function.

Performance Requirements - None.

Design Constraints - A single line may only contain one command. The next command must be on the next line.

Attributes - The "compiled" flag will be the code word "AB1E" in hexadecimal.

Other Requirements - Only Table 1 commands can be in the batch job since only those commands can be executed in a batch file. Super user commands (Table 2) must be sent manually to PANSAT by an operator.

(f) Default Batch Job. When the ground station is first implemented, it will include one premade batch job. This batch job will: connect to PANSAT, download the current telemetry, get the latest mail and file listings (LM N and LF N), post any mail that is in the mail message buffer, read then delete all mail to the ground station (RM E and DM E), then disconnect from PANSAT. These are the minimum functions to support the archive files and Internet site's operations.

(4) Set Super User Mode

Introduction - The control terminal in usually in the normal user mode. To enter the super user mode, the operator selects the super user option on the monitor program. The monitor is initially described in the control terminal section heading. The operator is then asked for a password. If the password is entered correctly, the terminal will be in super user mode for 10 minutes or until the mode is exited by the operator from the monitor, whichever comes first.

Inputs - The operator enters in the password.

Processing - When the operator requests super user mode, a dialog box querying for the password is displayed. If the correct password is entered, a timer is set for 10 minutes. The terminal is then put in

super user mode. While in super user mode, monitor display will show that fact. When the timer expires, normal mode is restored. If the operator manually turns off super user mode, the timer will be shut off. If the wrong password is entered the terminal beeps for a few seconds, then allow a retry.

Outputs - While in super user mode the top of the terminal display will display that the terminal is in super user mode. When normal mode is restored, the display will be removed. If the wrong password is entered, beeps are emitted from the terminal.

External Interfaces - The screen manager utilities are utilized by this function.

Performance Requirements - The ten minute timer is long enough to last for an entire PANSAT communication window.

Design Constraints - None.

Attributes - The password will be hard coded in the program. To change it, the program needs to be regenerated. The mode setting is a program flag.

Other Requirements - None.

(5) Archive Management Functions. This is identical to window described on the display terminal, except the functions are now located on the control terminal.

(6) Launch Other Applications

Introduction - This selection from the monitor program will allow the operator to launch another application from within the user services program. This new application will execute in tandem with user services, but each program will be completely independent of each other. This operation is similar to selecting the run option from the Windows 95/NT Start Menu, except it is all done from within the user services program.

Inputs - The operator chooses the application to launch.

Processing - A service call is made to the operating system to begin executing the application selected by the operator.

Outputs - None.

External Interfaces - The file system manager utilities are utilized by this function.

Performance Requirements - None.

Design Constraints - None.

Attributes - None.

Other Requirements - A typical application executed by this function would be to run the program which loads the spacecraft user services program onboard PANSAT.

c. Linux Terminal Functions.

The Linux terminal will not have a window interface. Rather all its functions will be performed in the background. As long as the terminal in on, these programs are required to be operational.

(1) Relay Communications Between PANSAT & Ground Station

Introduction - The terminal will continually wait to receive data from the satellite as well as any of the other ground station terminals. If data is received from the satellite, the data is transmitted to all the other three ground station terminals. If data is received a ground station terminal, it is relayed to the satellite. All the relayed data, going both directions, is stored for twenty-four hours on the local hard drive.

Inputs - Data from PANSAT is received via the modem. Data from a ground station terminal is received from the network via the socket datagram format.

Processing - The data is simply relayed and stored as it is received. If the ground station is not withing the satellite window of opportunity for communication, the Linux terminal will still attempt to send the data to PANSAT. The responsibility for keeping track of when to transmit is kept by the control and display terminals.

Outputs - The data for transmission is sent to PANSAT via the modem. The data received from PANSAT is broadcast to the other three ground station terminals on the network via the socket datagram format.

External Interfaces - The network and modem controller utilities are utilized by this function.

Performance Requirements - Because of the time constraints in communicating with PANSAT, the receive to transmit time lag for the relay function should be almost instantaneous.

Design Constraints - As the data is being relayed, it should simultaneously be saved on the local hard drive. No organization needs to be performed on the backup other than packet serialization.

Attributes - None.

Other Requirements - Once a day the backups of the daily transmitted packets should reviewed by the system. It should automatically purge any data packet older than twenty-four hours.

(2) Control Antenna Orientation

Introduction - This function is invisible to the operator. When the ground station is in PANSAT's footprint, the antenna is rotated to maintain the best possible reception with the satellite.

Inputs - The position of the satellite and the time the ground station are received from the display terminal via the network.

Processing - Once the ground station is within five minutes of being within PANSAT's footprint, the antenna is prepositioned for

the best reception. The best possible antenna position is based on PANSAT's provided position and the position of the NPS ground station. While in PANSAT's footprint, the antenna is updated with new positions as the satellite moves across the sky. Once PANSAT is beyond the horizon, this function stops updating the antenna until PANSAT's next revolution.

Outputs - The desired antenna position is sent to the antenna housing via the antenna driver.

External Interfaces - The network and antenna manager utilities are utilized by this function.

Performance Requirements - A new antenna position should be reevaluated every twenty seconds, which is the rate that new PANSAT positions are being supplied to this function.

Design Constraints - None.

Attributes - The NPS ground station position is a hard coded program constant.

Other Requirements - This function is contingent on a Windows 95/NT antenna driver being supplied by the client.

d. Server Functions.

Besides acting as the Windows NT network server, this terminal performs the following functions. These functions are performed in the background, requiring no explicit window interface.

(1) Maintain Archives. These underlying functions add the data received from PANSAT into the archives. These functions are automatic and should not be noticeable to the operator.

(a) Make Directory

Introduction - At the beginning of a new day, a new subdirectory is automatically created for that day's archived files.

Inputs - None.

Processing - When the internal clock switches to the new day, the directory is created. If the ground station is currently connected to PANSAT, the files will be stored in the last day's directory until the next connection.

Outputs - The make directory command is sent to the file system.

External Interfaces - The file system manager utilities are utilized by this function.

Performance Requirements - None.

Design Constraints - The file structure will look like:

..\ARCHIVE\year\month\day\

where year, month, and day will be filled in with the appropriate values.

Attributes - None.

Other Requirements - None.

(b) Store An Archive File

Introduction - This function is continually monitoring the data coming from PANSAT. If the data being received is telemetry, a mail message, a downloaded file, or a listing of mail/files, the data is captured and put into a archive file. The file is saved in the directory for the day it is captured.

Inputs - The data received from PANSAT is obtained via the SCC/modem drivers.

Processing - The data received from PANSAT is anticipated from the command that the operator sends to the satellite. When current telemetry is downloaded, after it is saved in a file, a signal is sent to the display telemetry data window on the display terminal. This signal gives the window the filename of the new telemetry data.

Outputs - The data received from PANSAT is saved on the hard disk via the file system. The signal is sent to the display terminal via the network.

External Interfaces - The network and file system manager utilities are utilized by this function.

Performance Requirements - This function needs to monitor the outgoing commands and incoming data at all times. No lag in processing can be permitted or data may be lost.

Design Constraints - For saving the archive files, the following scheme is used:

Table 4 - Archive file extensions.

Archive Data Type	File Extension
Current telemetry	.CT
Stored telemetry	.ST
Mail	.MAI
Files	.FIL
Mail listings	.MLI
File listings	.FLI

Attributes - None.

Other Requirements - As a mail message is being archived, the first line is checked. If the line is "Internet email:..." the mail message will be forwarded, via the Internet, to the email address specified after the colon.

(2) Maintain Internet Services. While the Internet services will be incorporated into a web site, these functions maintain the operations selected by the Internet users. This function will perform the interaction

between PANSAT and the web site and will ensure that the Internet users have the latest information to work with.

(a) Post mail to PANSAT

Introduction - When an Internet user enters in a mail message to be delivered to PANSAT, it is stored in a buffer. This function takes the buffer and puts it into the mail message buffer file. Thus, when the ground stations sends out its mail, the mail from the Internet site will be sent with it.

Inputs - The mail message(s) are retrieved from the Internet site's mail buffer via the network.

Processing - When this function detects the Internet site's mail buffer has a mail message in it, the message is taken and appended to the end of the mail message buffer (see the control terminal's send mail function). If the mail message buffer is already open, this function will wait until the buffer is closed, then it will add the Internet site's mail.

Outputs - The mail message(s) from the Internet site are sent to the file for the mail message buffer, via the file system.

External Interfaces - The network and file system manager utilities are utilized by this function.

Performance Requirements - The Internet site mail buffer only needs to be checked once an hour, then processed until all mail messages have been removed from the buffer.

Design Constraints - None.

Attributes - The name of the Internet site's mail buffer and the mail message buffer will be hard coded program constants.

Other Requirements - The system which will hold the Internet web site has not yet been identified by the client. For the sake of this function's design, however, it is assumed that the web site will be connected to the ground station via the network already in place between the control and display terminals.

(b) Retrieve A Mail Message From PANSAT

Introduction - If the user of the Internet web site requested a mail to be downloaded, a "read mail" command is inserted into the mail message buffer. During PANSAT's next pass, when the send outgoing mail is sent, the command requesting specific mail will also be sent.

Inputs - The mail number to retrieve is obtained from the Internet site's download buffer via the network.

Processing - When a message number is obtained from the Internet site's download buffer, the archive files are first checked to make sure the number is not already downloaded. The mail message buffer will then be checked to make sure the request has not already been processed. If the mail number or request is not in either place, a read mail command with the number requested is appended to the mail message buffer.

Outputs - The read mail command line is appended to the mail message buffer file via the file system.

External Interfaces - The network and file system manager utilities are utilized by this function.

Performance Requirements - The Internet site's download buffer will be checked once per hour, then processed until all requests are removed from the buffer.

Design Constraints - When a mail number is requested, only the most recent mail message with that number will be obtained. All previous mail messages with that number will have been deleted or overwritten by PANSAT.

Attributes - The name of the Internet site's download buffer and the mail message buffer will be hard coded program constants.

Other Requirements - The system which will hold the Internet web site has not yet been identified by the client. For the sake of this function's design, however, it is assumed that the web site will be connected to the ground station via the network already in place between the control and display terminals.

(c) Update HTML Pages For The Internet Site

Introduction - After a connection with PANSAT has ended, the most recently downloaded listing of mail, listing of files and the latest current telemetry values are converted into HTML pages and saved on the web site.

Inputs - The mail/file listings and telemetry values are obtained from the archive files via the file system.

Processing - The data from the archive files is put into a preformatted HTML mask. The filled out form is then sent to the Internet site as a web page, where it replaces the file listing, mail listing, and telemetry pages.

Outputs - The replacement web pages are sent to the Internet site via the network.

External Interfaces - The network and file system manager utilities are utilized by this function.

Performance Requirements - This function will be activated as soon as a disconnect command is sent to PANSAT by the ground station.

Design Constraints - None.

Attributes - The web page filenames will be hard coded program constants.

Other Requirements - The system which will hold the Internet web site has not yet been identified by the client. For the sake of this function's design, however, it is assumed that the web site will be connected to the ground station via the network already in place between the control and display terminals.

(3) Internet Virtual Site Functions. These functions are separate from the ground station software. Although they interact with the ground station, the components in this section are incorporated within an Internet web site, maintained on the server computer. This web site will be implemented using a combination of HTML pages and applets written in the JAVA programming language. The web site is hinged together with a main page. All of the functions listed in this section are branches from the main page.

(a) View PANSAT's Latest Directories

Introduction - This function is located on two pages. These two pages are updated by the server's maintain

Internet services function. One page will display the latest listings of the mail messages on PANSAT. The other page will display the latest listings of the files.

Inputs - The Internet user may select one of the mail messages.

Processing - From the mail page, the user can select a mail message. The selected mail message will be sent to the read a mail message function (see below). From either page the Internet user may go back to the main page or one of the other sub pages.

Outputs - The web pages are sent to the Internet user via the Internet.

External Interfaces - None.

Performance Requirements - None.

Design Constraints - None.

Attributes - None.

Other Requirements - None.

(b) View The Latest Telemetry From PANSAT

Introduction - This page simply displays the latest telemetry values obtained from PANSAT. This page is updated by the server's maintain Internet services function after every pass of the satellite.

Inputs - None.

Processing - The page is simply displayed. The Internet user may select to go back to the main page from here, or go to one of the other sub pages.

Outputs - The web page is sent to the Internet user via the Internet.

External Interfaces - None.

Performance Requirements - None.

Design Constraint - None.

Attributes - None.

Other Requirements - The actual values and format of the telemetry have not yet been defined by the client. These will need to be provided by the client before this function can be implemented.

(c) Read A Mail Message From PANSAT

Introduction - If a mail message was selected from the displayed mail listings option (above), it will be attempted to be retrieved from the archives. If a previous selection was not already made however, this page will prompt for a mail number to read, then attempt to retrieve it from the archives. If the mail number exists in the archives, it will be displayed. All displayed items will be viewable by the Internet user's browser program. If the mail message does not exist in the archives, however, the Internet user will be informed that the mail is not in the archives. The user will then be asked if they want to download the mail message during the next satellite pass.

Inputs - The mail number to display and the answer to the mail download query are entered by the Internet user, received via the Internet. The mail message, if it exists, is read from the archives files, via network and the file system.

Processing - Once the mail number has been obtained from the Internet user, the archive directories are searched through, starting with the current day's directory working backwards. The search will work back through the directories for the past 15 days, if need be. If the mail number has not been found after the 15 day search, the Internet user will be prompted if they want to attempt to download the mail message from PANSAT. If the answer is yes, the mail number is placed in the download buffer to

be handled by the control terminal. Otherwise, if the answer is no, the Internet user will be placed back on the main page.

Outputs - The mail message, if in the archives, will be sent to the Internet user via the Internet. The question to download may also be sent to the user.

External Interfaces - The network and file system manager utilities are utilized by this function.

Performance Requirements - None.

Design Constraints - The download buffer will be stored in a file.

Attributes - The filename for the download buffer will be a hard coded program constant.

Other Requirements - None.

(d) Post Mail On PANSAT

Introduction - This page will display a form for filling out a mail message. After the Internet user has completed entering in the mail message form, the mail will be sent to PANSAT during its next pass.

Inputs - The mail message is inputted by the Internet user via the Internet.

Processing - After the Internet user fills in the "from" line, "to" line, "subject" line (which is optional), and message body, the mail will be put into a mail message format. The "from" line will be replaced with the ground station's callsign, followed by the Internet user's identity in parenthesis. The mail message will then be placed in the mail buffer, to be processed by the control terminal of the ground station. The user will then be put onto the web site's main page.

Outputs - The web page is sent to the Internet user via the Internet.

External Interfaces - The file system manager utilities are utilized by this function.

Performance Requirements - None.

Design Constraints - The mail buffer will be stored in a file on the hard disk.

Attributes - The filename of the mail buffer will be a hard coded program constant.

Other Requirements - None.

(e) Display A Day's Archive File Directory. This function will behave identically as the "display a day's directory" function of the archive management window on the ground station (see display terminal functions above). The only difference is that it will be implemented as a web page, rather than a window feature. Additionally, the archive file directory will have to be accessed through the network as well as the file system management utilities.

(f) Search The Archive Files. This function will behave identically as the "search through archives function" of the archive management window on the ground station (see display terminal functions above). The only difference is that it will be implemented as a web page, rather than a window feature. Every option available in the window version will be available on web page. Additionally, the archive file directories will have to be accessed through the network as well as the file system management utilities.

(g) Download An Archive File

Introduction - Once a file has been selected in the "display a day's archive file directory" or "search the archive files" functions (the above two sections), this option can be selected. This will transfer the selected file to the Internet user.

Inputs - The selected file is obtained from the archives via the network and file system.

Processing - Using the standard Internet file transfer protocol, the selected file is sent to the Internet user. Once done, the Internet user is left in the directory or search listing functions they were previous in.

Outputs - The selected file is sent to the Internet user via the Internet.

External Interfaces - The network and file system manager utilities are utilized by this function.

Performance Requirements - None.

Design Constraints - The standard Internet file transfer protocol will operate very similarly to the download file operation with PANSAT.

Attributes - None.

Other Requirements - None.

V. SOFTWARE DESIGN

A. USE CASES

Use cases are analogous to a script of a play. A script describes the role of each actor in the play and how the actors behave in a scene. Similarly, use cases describe how each user of the system (actors) can perform in each possible instance. Unlike plays, however, use cases can be alternative and repetitive. Each use case describes a particular way of using the system in a goal-oriented manner. From the user's perspective, they precisely and concretely describe how the system can be used to achieve desired results.

Each use case involves an interaction between the end users, which will be referred to as actors in the use cases, and the system. Interactions may be complex and may involve other actors. Each use case potentially describes a large set of different interaction sequences. However, the different interaction sequences are analyzed in detail in the analysis and design phases.

Use cases can be hierarchically composed from parts. Use cases can not only have sub use cases, but can often share exceptions or sub activities. Using this structure makes the specification shorter and easier to understand. This hierarchy is represented in use case diagrams. The diamonds denote "parts of" relations and the labels indicated whether the parts are sub use cases, sub activities, or exceptions. Use cases are specifically activated by the system or called by their "parent" use case, as required. Activities are always ongoing, regardless of the system or "parent" use case's condition. However, activities with "parents" other than the system will begin executing only once their parent is activated. Exceptions are called whenever required to resolve an error condition.

The result of the use cases and diagrams is a description of the software's functional and dynamic behavior, which is a basis for the modeling in the system analysis phase [Awad pages 39 - 43].

1. Spacecraft Module Use Cases

The use cases for the spacecraft program are organized similar to a finite state automata. Data communication to and from a connection, or end user, is based on the state that the connection is in. Following that philosophy, the use cases work by tracing the data flow from when it arrives at the system through its various stages of processing. These stages of processing are determined according to the connection's status, which is a flag denoting its finite state. A few activities fall outside of the "follow the data flow" realm, but they still base their actions upon the connection status and/or program settings.

Use Case	(U1) data packet is received from SCOS
Actors	User or ground station.
Preconditions	A packet of data is received by PANSAT, it is delivered to user services by SCOS.
Description	When a data packet is delivered to user services from SCOS, this function first checks the originator of the packet. If the callsign is from an active connection, the packet is passed to (U20) for proper parsing. The only operation allowed by a non-connected end user is to connect. Thus, all packets from an un-connected callsign are routed to an appropriate connection process. If the callsign is the ground station, the packet is passed to (U13) to connect. Otherwise the packet is passed to (U14). Simultaneously, every packet that reaches the system is passed to (U21) for a signal strength evaluation.
Sub Use Cases	(U13) connect the ground station (U14) connect a user (U20) process a data packet from an active connection
Exceptions	(1) If the identity of the packet is undeterminable: ignore the packet.
Activities	(U21) check the signal strength of the incoming data packet
Postconditions	The incoming data packet is sent to the proper function for processing.

Use Case	(U2) perform general autodelete check
Actors	Autonomous activity of system.
Preconditions	Mail and/or files exist in storage.
Description	See page 71. Once a day, the time/date stamp of every mail and file in storage is checked. If the age of the normal mail/file (not sent to the "all" callsign) is greater than a threshold set by the ground station, delete it. Mail and files sent to "all" have their own threshold to compare, but if those are older than their threshold, they are deleted. If the additional autodelete settings are enabled and the amount of available memory in storage is equal to or less than the triggering amount, (U3) is activated.
Sub Use Cases	(U3) perform additional settings autodelete check
Postconditions	All mail and files which meet removal criteria have been removed.

Use Case	(U3) perform additional settings autodelete check
Actors	System use from of (U2).
Preconditions	Additional delete control setting is ON. Amount of space in storage is equal to or less than the threshold set in the control block. Mail and/or files exist in storage.
Description	See page 71. Similar to (U2), once per day, the system checks all the mail and files in storage. However, in this case the date/time stamp of the file is compared with up to three ground station mandated settings. For each one of the settings, if a file in storage is larger than or equal to the specified size and equal to or older than the age indicated in the setting, the file is deleted.
Exception	(1) If the control setting is ON, but no criteria settings are made: switch control setting to OFF, exit function.
Postconditions	All the mail and files which meet criteria will have been removed (in reality, only files will typically meet the criteria).

Use Case	(U4) perform interrupted file transfer record purge
Actors	Autonomous activity of system.
Preconditions	Connection records are stored in the interrupted file transfer list.
Description	See page 72. Once a day, the time/date stamp of every connection record that was put into the list is compared to a ground station set threshold. If the age of the connection record is equal to or greater than the threshold, it is purged from the list.
Postconditions	All mail and files which meet removal criteria have been removed.

Use Case	(U5) deadreckon satellite's position
Actors	Autonomous activity of system.
Preconditions	Deadreckon flag is set to ON.
Description	See page 74. Once a minute, the current position of the satellite is calculated. The calculation is based on deadreckoning from an initial position (which is periodically set by the ground station). This position is passed to (U6) and to (U8), which require the information for their operations.
Exceptions	(1) If the deadreckon flag is set to ON, but the initial position is not set: set the deadreckon flag to OFF. (U7) supplied position of PANSAT is out of date
Activities	(U6) control the cycling rate of modem (U8) defragment storage
Postconditions	The latest position of PANSAT is determined.

Use Case	(U6) control the cycling rate of modem
Actors	Activity of (U5).
Preconditions	Deadreckon flag is set to ON.
Description	When positional data is received from (U5), the position is compared to hard coded constants. These constants indicate when the footprint of the satellite is in areas where communication is not expected with PANSAT (such as above the ocean). When the satellite is in one of these areas of no-likely communications, the modem is put into a duty-cycling mode. When the footprint exits this area, the modem is put back in a normal mode.
Exceptions	(1) When there have been no communications with the ground station in 24 hours: disable the deadreckon flag and ensure the modem is in normal mode.
Postconditions	The modem is in either a duty-cycle or normal mode.

Use Case	(U7) supplied position of PANSAT is out of date
Actors	Exception of (U5).
Preconditions	The date of the PANSAT's basis position, supplied by the ground station, is older than 21 days, compared to the satellite's internal clock, which was also initialized by the ground station.
Description	The next time the ground station is connected to the satellite, it is sent a message warning that the positional data is out of date. The deadreckon and defragment flags are set to OFF.
Postconditions	The deadreckon and defragment flags are set to OFF.

Use Case	(U8) defragment storage
Actors	Activity of (U5).
Preconditions	The defragment and deadreckon flags are set to ON.
Description	See page 75. When positional data is received from (U5), the position is compared to hard coded constants. These constants indicate when the footprint of the satellite is in areas where communication is not expected with PANSAT (such as above the ocean). When PANSAT enters this area, it determines the fragmentation value of the storage. If the overall file fragmentation percentage is below the ground station set threshold, then the files are defragmented, one file at a time, until the area of no likely communications is exited. Next time that area is entered, the process repeats.
Exceptions	(1) If the defragment flag is ON, but the deadreckoning flag is OFF: set defragment flag to OFF.
Postconditions	After each time activated, at least one file is defragmented.

Use Case	(U9) check connection for autodisconnect
Actors	Autonomous activity of system.
Preconditions	One user or ground station is currently connected.
Description	See page 72. Once per minute all the active connection records are checked. If the time the last input was received from a connection is over the threshold (set by ground station), the user is disconnected (U64). If the user was in the middle of a file transfer, (U10) is called to save the connection information in order to continue the file transfer later.
Sub Use Cases	(U10) save user's file-transfer status (U64) disconnect
Postconditions	All idle connections are disconnected, freeing up their connection records.

Use Case	(U10) save user's file-transfer status
Actors	System use from (U9).
Preconditions	A user about to be disconnected is still conducting a file transfer.
Description	If the connection status is "temp uploading/downloading", the record currently in the list for this connection is updated. Otherwise, the record of the file transfer is added to the list. If the list already has the maximum number, the connection is not saved, but erased. When the user reestablishes the connection, the information from the list is used to continue the file transfer exactly where it left off. See page 38 for the attributes to be saved.
Exceptions	(1) If the number of connection records stored exceeds the number set by ground station: abort operation.
Postconditions	All the necessary file transfer information for the user is saved.

Use Case	(U11) compress data.
Actors	Autonomous activity of the system
Preconditions	Data is ready to be passed to the AX.25 packet utility in preparation for transmission to an end user or data is ready to be passed to the Surrey file system for saving in storage.
Description	See page 76. If the data is going to be sent to an NPSterm connection, the data is compressed before it is sent to the AX.25 utilities. If the data compression flag is set to ON and the data is going to be saved in storage, then the data is compressed. Otherwise, the data is left unmodified.
Exception	(1) If the data is already compressed: do not compress again, skip the function. The data is known to be compressed by a marker saved in the file information data.
Postconditions	The data is ready to packeted by the AX.25 utilities or saved in storage.

Use Case	(U12) decompress data.
Actors	Autonomous activity of the system
Preconditions	Data has been received from the AX.25 packet utilities that originated from an end user or data is retrieved from the Surrey file system.
Description	See page 78. If the packet coming from the AX.25 utilities originated from an NPSterm connection, the data is decompressed upon arrival at PANSAT. If the file information for the data retrieved from storage denotes a compressed file, the data is also decompressed. However, two special cases to skip decompression exist. The first case is if the data coming from an NPSterm is going directly into storage and the data compression flag is ON. The second case, in transmitting a file from PANSAT, is if the file information indicates the file is compressed and the file is to be sent directly to an NPSterm connection. In either case, the data is not decompressed, since if it was, it would only be compressed again immediately after.
Exception	(1) The information indicates that the data should be compressed, but the data is not: pass the data through with no modification or error raising.
Postconditions	The data is uncompressed, ready to use or manipulate.

Use Case	(U13) connect the ground station
Actors	Ground station from (U1).
Preconditions	The ground station is not yet connected and a packet is received from the ground station's callsign.
Description	See page 33. If the ground station sends the "request to connect" command, the ground station's connection record is set to "active" and updated, setting the connection status to "connecting". The time and date of the connection attempt is sent to the log manager utility. Control is then passed to (U16).
Sub Use Cases	(U16) display the greeting message
Exceptions	(1) The packet does not contain the command "C PANSAT": abort the connection process and ignore the packet. (2) The ground station tries to connect, but a connection with the ground station is already ongoing: deny the second connection and log the failed attempt.
Postconditions	The ground station is connected and the connection record is updated.

Use Case	(U14) connect a user
Actors	User from (U1).
Preconditions	A packet is received from a callsign of a user who is not currently connected.
Description	See page 33. If there is a connection record available (there is a maximum of 15), the available record is marked as active and filled out, including setting the connection status to "connecting". Control is then passed to (U16). If there are no available connection records or the "terminate user services" process has commenced, then control is passed to (U15).
Sub Use Cases	(U16) display the greeting message
Exceptions	(1) The packet does not contain command "C PANSAT": abort the connect process and ignore the packet. (U15) connection request is denied
Postconditions	The user is connected and the connection record is updated.

Use Case	(U15) connection request is denied
Actors	Exception of (U11).
Preconditions	A user has tried to connect, but PANSAT cannot allow the connection.
Description	If the reason for the denied connection is all connection records are active, the user is sent a message saying "All connections are occupied, please try again in one minute". If the "terminate user services" process has commenced however, the user is sent "System temporarily unavailable - disconnect now". The control thread for this particular packet terminates after the proper warning has been sent.
Postconditions	The connection process terminates without a new connection being established.

Use Case	(U16) display the greeting message
Actors	User from (U13) or ground station from (U14).
Preconditions	A connection has just been established with PANSAT.
Description	See page 35. "Welcome to PANSAT" is sent to the connection. If the connection is the ground station, it is sent the time and date of the last ground station connection. Next, a happy face is sent to the end user. The happy face is made out of simply telemetry values. Finally, the "time of last input from user" field in the connection record is set to the current time. Control is then passed to (U17).
Sub Use Cases	(U17) check if callsign is in the list of interrupted file transfers
Postconditions	A greeting message has been sent to the end user.

Use Case	(U17) check if callsign is in the list of interrupted file transfers
Actors	User or ground station from (U16).
Preconditions	A connection has just been established with PANSAT and has been sent a greeting message.
Description	See page 37. The callsign of the connection is compared to the callsigns in the interrupted file transfer list. If one of the callsigns matches, the end user is queried "Continue previous file transfer?". The control thread is then terminated for this packet (the end user's response will be properly parsed, identified by the connection status). Otherwise, control is directly passed to (U18).
Sub Use Cases	(U18) display the broadcast message
Postconditions	If the end user was previously interrupted in a file transfer, they have the opportunity to continue the file transfer exactly where they left off.

Use Case	(U18) display broadcast message
Actors	Ground station or user from (U17), (U23), (U24), or (U26).
Preconditions	All other steps in the connection process are complete. Connection status is "connecting".
Description	See page 38. The broadcast message (set by the ground station) is sent to the user or ground station, then control is passed to (U19).
Sub Use Cases	(U19) display the menu options
Postconditions	The broadcast message has been sent to the user/ground station.

Use Case	(U19) display the menu options
Actors	Ground station or user from (U18), (U22), (U23), (U24), (U29), (U30), (U31), (U32), (U34), (U35), (U36), (U37), (U38), (U42), (U43), (U44), (U45), (U46), (U47), (U48), (U49), (U51), (U53), (U54), (U56), (U59), (U60), (U62), (U63), (U65) or (U66).
Preconditions	The system is ready to process a command from the user or ground station.
Description	See page 40. A prompt is sent to the user or ground station with all the menu options. The connection status is set to "menu". This control thread is now finished and the connection is waiting for a command from the user or ground station.
Postconditions this.	The connection is ready to receive a BBS command and end user has been notified of

Use Case	(U20) process a data packet from an active connection
Actors	Ground station or user from (U1).
Preconditions	A packet has been received by PANSAT from either a user or the ground station whom is currently connected.
Description	This function merely directs the incoming packet to the correct sub-function, based solely on the connection status value for the particular callsign. The connection record's "time of last input from the user" field is updated with PANSAT's current time. The connection must be in one of seven possible statuses, with the matching sub use case being passed the control.
Sub Use Cases	(U23) Status: uploading or temp uploading (U24) Status: downloading or temp downloading (U25) Status: connecting (U27) Status: verifying (U28) Status: menu (U29) Status: sending broadcast (U30) Status: sending mail
Exceptions	(1) If the connection isn't in one of the seven states, send a warning to user of "Software fault-status error", then disconnect the user by executing (U64).
Postconditions	Control is passed to the proper function to process the packet from the user/ground station.

Use Case	(U21) check the signal strength of the incoming data packet
Actors	Ground station or user from (U1)
Preconditions	A packet is received by the SCOS.
Description	See page 79. All packets received by PANSAT are sent to this function, along with the signal strength of the transmission carrying the packet. If the power of the signal received is greater than a program constant threshold, the end user sending the signal is warned of the overbearing power. If three more packets are received from the same user without the power being reduced, the user is disconnected.
Sub Use Cases	(U64) disconnect
Postconditions	Either the user's power is within power specifications or an offending user has been disconnected from PANSAT.

Use Case	(U22) file size over threshold
Actors	Exception from (U23).
Preconditions	The latest data packet received from the user during a file upload, when added to the file, makes the file larger than the threshold size set by the ground station.
Description	The file upload is terminated. A message is sent to the user telling them the file size limit was exceeded. The file is deleted from the system. Control is then transferred to (U19).
Sub Use Cases	(U19) display the menu choices
Postconditions	The file which would have been too large is removed from the system.

Use Case	(U23) status: uploading or temp uploading
Actors	Ground station or user from (U20).
Preconditions	The file has been created and is open in PANSAT's storage.
Description	All the data received from the connection is merely directed into the open file. When the end-of-file character is received, the file is closed. After which, a message is sent to the user stating that the file transfer is complete and how much memory the file took up. Next, the control thread is passed. If the connection status is "temp uploading", control is switched to (U18). Otherwise, control is passed to (U19). After each packet received is added to the file, the file size is checked. If the restrict mail/file size option is set and the file is over the maximum (both value set by the ground station) , (U22) is executed.
Sub Use Cases	(U18) display broadcast message (U19) display the menu options
Exceptions	(U22) file size over threshold
Postconditions	The file is uploaded and saved in PANSAT's storage.

Use Case	(U24) status: downloading or temp downloading
Actors	Ground station or user from (U20).
Preconditions	While PANSAT is downloading to a connection, that connection transmits a data packet.
Description	Since a connection that is receiving a file should only be able to transmit acknowledgments, this action is deemed as a user abort of the download. The file transfer is ceased and the message "Downloading interrupted by user" is sent to the end user. If the connection status was "temp downloading", control is passed to (U18). Otherwise, control is passed to (U19).
Sub Use Cases	(U18) display broadcast message (U19) display the menu options
Postconditions	The file transfer is canceled by the user/ground station.

Use Case	(U25) status: connecting
Actors	Ground station or user from (U20).
Preconditions	In their last connection, a user or the ground station was interrupted during a file transfer. The connection was sent a query asking if it wants to continue the file transfer.
Description	If the packet from the end user contains an affirmative answer: if the file transfer was uploading, the destination file is open for append and the connection status is set to "temp uploading". However, if the file transfer was downloading, the file being copied is opened with the file pointer placed where the transfer left off, then the connection status is set to "temp downloading". In either case, the control thread is terminated as further data from the connection will be appropriately handled. If the file transfer is to be skipped (the answer from the end user was negative), control is passed to (U18). No matter what the user answered, once control leaves this function, the corresponding connection record is removed from the interrupted file transfer list. If neither a discernable yes or no was received, exception (U26) is raised.
Sub Use Cases	(U18) display broadcast message
Exceptions	(U26) illegal command received
Postconditions	If the user/ground station desired it, the file transfer has been completed.

Use Case	(U26) illegal command received
Actors	Exception from (U25) or (U28).
Preconditions	A response that PANSAT expected from the ground station or user was not the one received.
Description	The user is notified which one of the three errors is committed: an option sent is not available, an option sent is restricted from that user, or the parse of a legal command sent failed - it was in a wrong format (the portion that failed is shown to the user).
Postconditions	The connection status is unchanged, and control is passed back to the function that called this exception, where it is waiting for a correct response.

Use Case	(U27) status: verifying
Actors	Ground station from (U20).
Preconditions	The ground station sent a restricted command for PANSAT. The command line received was placed in the special command buffer and the ground station was sent a verification query.
Description	See page 69. If the packet received from the ground station contains the correct answer to the verification query sent in (U55), (U56) or (U60), then the special command buffer is checked. Control is then passed to one of the six sub cases, based on the command line in the buffer. However, if the verification answer was wrong, exception (U32) is raised.
Sub Use Cases	(U33) process a "post broadcast" command (U34) process a "get BBS settings" command (U35) process a "ground station delete mail" command (U36) process a "ground station delete file" command (U37) process a "update ground station callsign" command (U39) process a "terminate user services program" command (U42) process a "update BBS settings" command
Exceptions	(U32) incorrect verification response received
Postconditions	The ground station connection is allowed to continue or it is terminated.

Use Case	(U28) status: menu
Actors	Ground station or user from (U20).
Preconditions	Connection status is "menu" and the system is ready to accept a command from the user/ground station.
Description	The command line sent by the user or ground station is parsed, obtaining the fields as shown in the syntax column of Tables 1 and 2 (see page 24). If the command line parses without errors, the parsed fields are sent as parameters to the appropriate one of the twenty-two commands listed as sub use cases. Control is then passed to this sub use case. If the command doesn't parse or the command is not allowed for a user, exception (U28) is raised.
Sub Use Cases	(U43-U47), (U49), (U51-U52), (U54-56), (U58-U60) or (U62-U66) is appropriately selected based on the command sent by the user or ground station.
Exceptions	(U26) illegal command received
Postconditions	The command from the user/ground station has been parsed and sent to the appropriate function.

Use Case	(U29) status: sending broadcast
Actors	Ground station from (U20).
Preconditions	The ground station is in the process of updating the broadcast message. The verification function was correctly answered and broadcast file is open.
Description	Until the end-of-message character is reached, all data received from the ground station is put into the broadcast file. When the end-of-message character is reached, the broadcast file is closed and control is switched to (U19).
Sub Use Cases	(U19) display the menu options
Exceptions	(1) If a user (not the ground station) is in this connection status: the message "Software fault-not super user" is sent to the user, then control is switched to (U19).
Postconditions	The broadcast message has been replaced with a new message.

Use Case	(U30) status: sending mail
Actors	Ground station or user from (U20)
Preconditions	The user/ground station is in the process of sending a mail message. The file containing the mail message is open and waiting for data.
Description	Until the end-of-message character is reached, all data received from the user or ground station is placed in the mail file. When the end-of-message character is reached, the mail file is closed and control is switched to (U19). After each packet received is added to the mail message, the mail size is checked. If the restrict mail/file size option is set and the mail messsage is over the maximum (both value set by the ground station) , (U31) is executed.
Sub Use Cases	(U19) display the menu options
Exceptions	(U31) mail size over threshold
Postconditions	The mail message is saved and available in PANSAT's storage for the ground station or any user.

Use Case	(U31) mail size over threshold
Actors	Exception from (U30).
Preconditions	The latest data packet received from the user during the sending of mail, when added to the mail message, makes the mail larger than the threshold size set by the ground station.
Description	Sending of mail is terminated. A message is sent to the user telling them the mail size limit was exceeded. The mail is deleted from the system. Control is then transferred to (U19).
Sub Use Cases	(U19) display the menu choices
Postconditions	The file which would have been too large is removed from the system.

Use Case	(U32) incorrect verification response received
Actors	Exception of (U27).
Preconditions	The ground station has sent the wrong answer to the verification query.
Description	The ground station is sent the warning "Verification incorrect - access denied". The time and date, and command attempted are sent to the log manager. Control is then passed to (U19).
Sub Use Cases	(U19) display the menu options
Postconditions	The command attempted is aborted and control is back at the menu.

Use Case	(U33) process a "post broadcast" command
Actors	Ground station from (U27).
Preconditions	The proper command is in the special command buffer and the correct verification answer was received.
Description	See page 60. The file containing the broadcast message is opened, overwriting the old message. The connection status is set to "sending broadcast". The control thread is then terminated. The next packets from the ground station will be properly parsed.
Postconditions	The broadcast message file is open and ready to receive the new message.

Use Case	(U34) process a "get BBS settings" command
Actors	Ground station from (U27)
Preconditions	The proper command is in the special command buffer and the correct verification answer was received.
Description	See page 61. The control setting values are sent to the ground station in a compact format, the break out of which is described in Table 3. When done, control is switched to (U19).
Sub Use Cases	(U19) display the menu options
Postconditions	The control settings are passed from PANSAT to the ground station.

Use Case	(U35) process a "ground station delete mail" command
Actors	Ground station from (U27).
Preconditions	The proper command is in the special command buffer and the correct verification answer was received.
Description	See page 48. For every single mail specified in the command line, unconditionally remove it from PANSAT's storage. If a mail attempting to be deleted is not on the system, call exception (U61). Once done, control is passed to (U19).
Sub Use Cases	(U19) display the menu options
Exception	(U61) mail not found
Postconditions	All the mail specified by the ground station have been removed from the system.

Use Case	(U36) process a "ground station delete file" command
Actors	Ground station from (U27).
Preconditions	The proper command is in the special command buffer and the correct verification answer was received.
Description	See page 50. For every single file specified in the command line, unconditionally remove it from PANSAT's storage. If a file attempting to be deleted is not on the system, exception (U50) is called, then the de[illegible]n attempts are continued with the next specified file. Once done, control is pass[illegible].o (U19).
Sub Use Cases	(U19) display the menu options
Exception	(U50) file not found
Postconditions	All the files specified by the ground station have been removed from the system.

Use Case	(U37) process a "update ground station callsign" command
Actors	Ground station from (U27).
Preconditions	The proper command is in the special command buffer and the correct verification answer was received.
Description	See page 66. The callsign for the ground station is replaced with the one parsed in the command line. This callsign is repeated back to the ground station for a non-active verification, after which control is passed to (U19). If any of the repeated new callsign values differ, exception (U38) is raised.
Sub Use Cases	(U19) display the menu options
Exceptions	(U38) redundant data received does not match
Postconditions	The callsign for the ground station is updated and will be in affect for the ground station's next connection.

Use Case	(U38) redundant data received does not match
Actors	Exception from (U37) or (U42)
Preconditions	The repeated values passed to PANSAT do not match.
Description	This indicates an error in transmission occurred while PANSAT was receiving the packet. Since the information in the packet is critical (software settings), the packet is discarded. The ground station is sent the warning "Error in critical packet, command aborted". Control is then passed to (U19).
Sub Use Cases	(U19) display the menu options
Postconditions	The callsign for the ground station is updated and will be in affect for the ground station's next connection.

Use Case	(U39) process a "terminate user services program" command
Actors	Ground station from (U27).
Preconditions	The proper command is in the special command buffer and the correct verification answer was received.
Description	See page 67. A flag indicating that user services is terminating is set to ON (this disables further connections with PANSAT). (U40) is called to let all the users connected to the satellite of the pending shutdown. The system then waits one minute. Then (U41) is called to disconnect every user currently connected to the PANSAT. Finally, all files are closed, all dynamic memory is deallocated, and the program exits out to the operating system.
Sub Use Cases	(U40) warn all users (U41) disconnect all users
Postconditions	The user services program has gracefully exited to the operating system.

Use Case	(U40) warn all users
Actors	Ground station from (U39).
Preconditions	The "terminate user services" process has commenced.
Description	If any connections are conducting file transfers, the operation is terminated. Then the message "System will temporarily shut down in one minue - disconnect now" will be sent to every user currently connected to PANSAT. Control then returns to the calling function.
Postconditions	Every user connected to PANSAT has been warned of the user services shutdown.

Use Case	(U41) disconnect all users
Actors	Ground station from (U39).
Preconditions	The "terminate user services" process has commenced and one minute has passed since all connected users were warned.
Description	Every single connection record is set to inactive, then control is returned to the calling function.
Postconditions	The control settings are updated.

Use Case	(U43) get current telemetry.
Actors	Ground station or user from (U28).
Preconditions	The proper command has been received as per Table 1.
Description	See page 40. The telemetry values in a formatted mask is sent to ASCII terminal. If the connection is an NPSterm, then the telemetry values are sent in a concatenated list, letting the terminal software format it. When done, control is returned to (U19).
Sub Use Cases	(U19) display the menu options
Postconditions	The telemetry at the instant the command was received is delivered to the user/ground station.

Use Case	(U44) get stored telemetry
Actors	Ground station or user from (U28).
Preconditions	The proper command has been received as per Table 1.
Description	See page 41. First the telemetry file is opened and the connection status is set to "downloading". Next, the contents of the open file are copied to the user, packet by packet, until done. After each packet is sent to the user, the file position pointer in the connection record is updated. When entire file has been copied, the file is closed and control is switched to (U19).
Sub Use Cases	(U19) display the menu options
Postconditions	The file containing the telemetry values for the past few days has been downloaded.

Use Case	(U45) switch to ASCII
Actors	Ground station or user from (U28).
Preconditions	The proper command has been received as per Table 1 and the end user is switching out of NPSterm.
Description	See page 57. The connection record is updated to indicate ASCII interface will now be used. From now on normal character data streams and no data compression will be used in communicating with the connection. Control is then switched to (U19).
Sub Use Cases	(U19) display the menu options
Postconditions	The connection is in ASCII mode.

Use Case	(U46) switch to NPSterm
Actors	Ground station or user from (U28).
Preconditions	The proper command has been received as per Table 1 and the end user is using NPSterm software.
Description	See page 55. The connection record is updated to indicate that NPSterm is being used. The version number of NPSterm is sent to user or ground station (so the mode switch can be canceled if need be). From now on, shortcuts and data compression will be used in communicating with the connection. After the update, control is switched to (U19).
Sub Use Cases	(U19) display the menu options
Postconditions	The connection is in NPSterm mode.

Use Case	(U47) send one-liner
Actors	Ground station or user from (U28)
Preconditions	The proper command has been received as per Table 1.
Description	See page 57. The one line message parsed from the command line is immediately sent to the callsign indicated. The only exception to this rule is if the callsign is using an ASCII terminal and is conducting a file transfer. In that case, the message is sent to the callsign as soon as the file transfer completes. If the callsign indicated is not currently connected to PANSAT, control is switched to (U48). If "all" is used instead of a callsign, every user connected to PANSAT at the instant the command is received will be sent the message. When the process is completed, control is passed to (U19).
Sub Use Cases	(U19) display the menu options
Exceptions	(U48) user not connected
Postconditions	A one line message has been sent to the specified user.

Use Case	(U48) user not connected
Actors	Exception from (U47).
Preconditions	A one-liner was attempted to be sent to a callsign not currently connected to PANSAT.
Description	The originator of the one line message is sent the warning, "callsign not currently connected to PANSAT, message not sent." Control is then passed to (U19).
Sub Use Cases	(U19) display the menu options
Postconditions	The user or ground station is at command prompt, but the one-liner message was not sent.

Use Case	(U49) read a file
Actors	Ground station or user from (U28).
Preconditions	The proper command has been received as per Table 1.
Description	See page 47. First the file identified by the number in the command line is opened and the connection status is set to "downloading". Next, packet by packet, the contents of the file are copied to the user, until the end of file is reached. After each packet is sent to the user, the file position pointer is updated in the connection record. When the copying process is completed, control is passed to (U19). If the file number specified does not exist in PANSAT's storage, the user is sent an interrogative packet to stop them from downloading. Then exception (U50) is raised before passing control to (U19).
Sub Use Cases	(U19) display the menu options
Exceptions	(U50) file not found
Postconditions	The desired file has been sent to the user.

Use Case	(U50) file not found
Actors	Exception from (U36), (U49), (U51), or (U56).
Preconditions	The file number requested does not exist in PANSAT's storage.
Description	A warning "File # does not exist" is sent to the user or ground station, then control is returned to the use case raising the exception to handle the error condition.
Postconditions	Control is returned to the function which called this exception.

Use Case	(U51) forward file
Actors	Ground station or user from (U28).
Preconditions	The proper command has been received as per Table 1.
Description	See page 55. If the file number specified in the command line exists, all the callsigns parsed from the command line are added to the file's "still to" list. Additionally, a new forward line is also added to the file information, with each of the newly specified callsigns. Control is then transferred to (U19). If the file number does not exist, however, exception (U50) is raised before transferring control to (U19).
Sub Use Cases	(U19) display the menu options
Exceptions	(U50) file not found
Postconditions	The file has been forwarded/sent to new callsigns.

Use Case	(U52) send file
Actors	Ground station or user from (U28)
Preconditions	The proper command was received as per Table 1.
Description	See page 44. First the file system is checked to ensure that the filename is not already being used in storage. If the filename is already in use, control is switched to exception (U53). Otherwise, the connection status is set to "uploading". Then the next available file number is assigned to the file and the file is created - opening it, ready for input. A second file is created, the "information file". This second file contains the "to", "from", "still to", and "filename" information, obtained from the command line. The second file is closed. The control thread is terminated as all further input from the connection will be properly parsed and directed to the open file until the end of file is reached.
Exceptions	(U53) filename already exists
Postconditions	The file is open and ready to receive data.

Use Case	(U53) filename already exists
Actors	Exception from (U52).
Preconditions	While creating a new file, a file already exists in PANSAT's storage with the same filename.
Description	The warning "Filename already exists, command aborted" is sent to the user or ground station, then control is transferred to (U19). This error is raised to prevent duplicate files from being stored to the memory-limited satellite.
Sub Use Cases	(U19) display the menu options
Postconditions	The filename was not duplicated and the user was put back at the "command prompt."

Use Case	(U54) list files
Actors	Ground station or user from (U28).
Preconditions	The proper command has been received as per Table 1.
Description	See page 52. If the "N" option is used, all files posted within the last 24 hours are listed. Otherwise, the files with numbers matching the range specified in the command line are listed to the user. The default is to list all files in the range, but if the command line has the flag "A", all the files without the end user's callsign in the "still to" list are not listed out, except if the files were sent to "all". If the flag "U" was used, files are restricted from being listed just as with the "A" option, but the files sent to "all" are restricted as well. If no file numbers are in the range specified in the command line, an error is not raised, rather nothing is listed. After the list has been sent to the user or ground station, control is passed to (U19).
Sub Use Cases	(U19) display the menu options
Postconditions	The list of files has been supplied to the user/ground station.

Use Case	(U55) ground station only command received.
Actors	Ground station from (U28).
Preconditions	One of the commands listed in Table 2 (except the delete commands - they are handled in the regular delete use cases) was received from the ground station.
Description	See page 69. The restricted command being requested is put into the special command buffer and the connection status is set to "verifying". For a verification query, a series of random numbers are sent to the ground station. The control thread is terminated, as the correct response will be properly parsed by the system.
Postconditions	The command sent up by the ground is prepared to be executed, following receipt of the correct answer to the verification query.

Use Case	(U56) delete a file
Actors	Ground station or user from (U28).
Preconditions	The proper command has been received as per Table 1.
Description	See page 50. If the command line ends with a "SU" and the originator of the command is the ground station, then the command line is entered into the special command buffer and the connection status is set to "verifying". The ground station is sent a verification query (set of random numbers), then control for the thread terminates (waiting for the correct answer reply). Otherwise, if no "SU" flag is set, a temporary list is made out of every file number specified in the command line. This list is processed sequentially, examining each file to be deleted separately. If the end user was the originator of the file attempting to be deleted, the file is removed from the system. If the end user was a recipient of the file, the end user's callsign is removed from the "still to" list (if "still to" list becomes empty, then the file is removed from system). If the end user was neither the originator or a recipient, exception (U57) is called, then the next number in the list is processed. If the file number attempting to be deleted does not exist, exception (U50) is called, then the next number in list is processed. When every number in the list has been processed, control is switched to (U19).
Sub Use Cases	(U19) display the menu options
Exceptions	(U50) file not found (U57) no privilege to delete
Postconditions	All the specified files have been deleted.

Use Case	(U57) no privilege to delete
Actors	Exception from (U56) or (U60).
Preconditions	A user has tried to delete mail or a file for which they were neither the originator nor recipient.
Description	A message is sent to the user warning that the deletion was not allowed for that number.
Postconditions	Control is returned to the function which called this exception.

Use Case	(U58) send mail
Actors	Ground station or user from (U28).
Preconditions	The proper command has been received.
Description	See page 42. First a file is created and left open for the mail using a unique mail number. The file is started with the "to", "from", "still to" and "subj" lines, filled out with the values parsed from the command line (the "still to" is initialized to be identical to the "to" line). The connection status is set to "sending mail" and the control thread is terminated.
Postconditions	The file is open and ready to receive the mail message.

Use Case	(U59) list mail
Actors	Ground station or user from (U28).
Preconditions	The proper command has been received as per Table 1.
Description	See page 50. If the "N" option is used, all mail posted within the last 24 hours are listed. Otherwise, the mail with numbers matching the range specified in the command line are listed to the user. The default is to list all mail in the range, but if the command line has the flag "A", all the mail without the end user's callsign in the "still to" list are not listed out, except if the mail was sent to "all". If the flag "U" was used, mail are restricted from being listed just as with the "A" option, but the mail sent to "all" are restricted as well. If no mail numbers are in the range specified in the command line, an error is not raised, simply nothing is listed. After the list has been sent to the user or ground station, control is passed to (U19).
Sub Use Cases	(U19) display the menu options
Postconditions	The list of mail has been supplied to the user/ground station.

Use Case	(U60) delete mail
Actors	Ground station or user from (U28).
Preconditions	The proper command has been received as per Table 1.
Description	See page 48. If the command line ends with a "SU" and the originator of the command is the ground station, then the command line is entered into the special command buffer and the connection status is set to "verifying". The ground station is sent a verification query (set of random numbers), then control for the thread terminates (waiting for the correct answer reply). Otherwise, if no "SU" flag is set, a temporary list is made out of every mail number specified in the command line. This list is processed sequentially, examining each mail to be deleted separately. If the end user was the originator of the mail attempting to be deleted, the mail is removed from the system. If the end user was a recipient of the mail, the end user's callsign is removed from the "still to" list (if "still to" list becomes empty, then the mail is removed from system). If the end user was neither the originator or a recipient, exception (U57) is called, then the next number in the list is processed. If the mail number attempting to be deleted does not exist, exception (U61) is called, then the next number in list is processed. When every number in the list has been processed, control is switched to (U19).
Sub Use Cases	(U19) display the menu options
Exceptions	(U61) mail not found (U57) no privilege to delete
Postconditions	All the specified mail have been deleted.

Use Case	(U61) mail not found
Actors	Exception from (U35), (U60), (U62), or (U66).
Preconditions	The mail number requested does not exist in PANSAT's storage.
Description	A warning "Mail # does not exist" is sent to the user or ground station, then control is returned to the use case raising the exception to handle the error condition.
Postconditions	Control is returned to the function which called this exception.

Use Case	(U62) read mail
Actors	Ground station or user from (U28).
Preconditions	The proper command has been received as per Table 1.
Description	See page 46. The file number designated is opened and the connection status is set to "downloading". The contents of the file are copied to the end user. When the end of message is reached, the file is closed. If the mail number doesn't exist, however, exception (U61) is called. After each case, control is transferred to (U19).
Sub Use Case	(U19) display the menu options.
Exceptions	(U61) mail not found
Postconditions	The desired mail was sent to the user.

Use Case	(U63) help
Actors	User from (U28) (the ground station will not send this command or any user using NPSterm, as NPSterm will intercept the command and provide a more robust help feature).
Preconditions	The proper command has been received as per Table 1.
Description	See page 53. All the commands and their syntax (the first and third columns of Table 1) are listed out to the user, then control is switched to (U19).
Sub Use Cases	(U19) display the menu options
Postconditions	The user has been given help, but is that what they really need?

Use Case	(U64) disconnect
Actors	Ground station or user from (U28), or system from (U9), or (U21).
Preconditions	The callsign specified is connected.
Description	See page 59. The connection record for the callsign is updated as being inactive, making it available for another connection.
Postconditions	The user or ground station is disconnected and must reconnect to communicate with PANSAT.

Use Case	(U65) who
Actors	Ground station or user from (U28).
Preconditions	The proper command has been received as per Table 1.
Description	See page 52. The callsign of each connection currently active is listed to the requesting user or ground station, then control is switched to (U19).
Sub Use Cases	(U19) display the menu options
Postconditions	The callsigns of all the users currently connected to PANSAT are listed out.

Use Case	(U66) forward mail
Actors	Ground station or user from (U28).
Preconditions	The proper command has been received as per Table 1.
Description	See page 54. If the mail number specified in the command line exists, all the callsigns parsed from the command line are added to the mail's "still to" list. Additionally, a new forward line is also added to the beginning of the file, with each of the newly specified callsigns. Control is then transferred to (U19). If the mail number does not exist, however, exception (U61) is raised before transferring control to (U19).
Sub Use Cases	(U19) display the menu options
Exceptions	(U61) mail not found
Postconditions	The mail has been forwarded/sent to new callsigns.

2. Spacecraft Module Use Case Diagram

Due to the size and complexity of this diagram, it has been split into five parts. The use cases boxes are color coded: a blue box appears only once throughout the diagram parts, a red box appears multiple times (but all of the instances should be considered as just a single instance), and a green box also appears multiple times, but it is used as a hinge between the diagram parts (i.e. if space permitted, the entire Part II diagram would be placed inside Part I, connected with U1). Additionally, the PANSAT User Services System as a whole is denoted by a pink box.

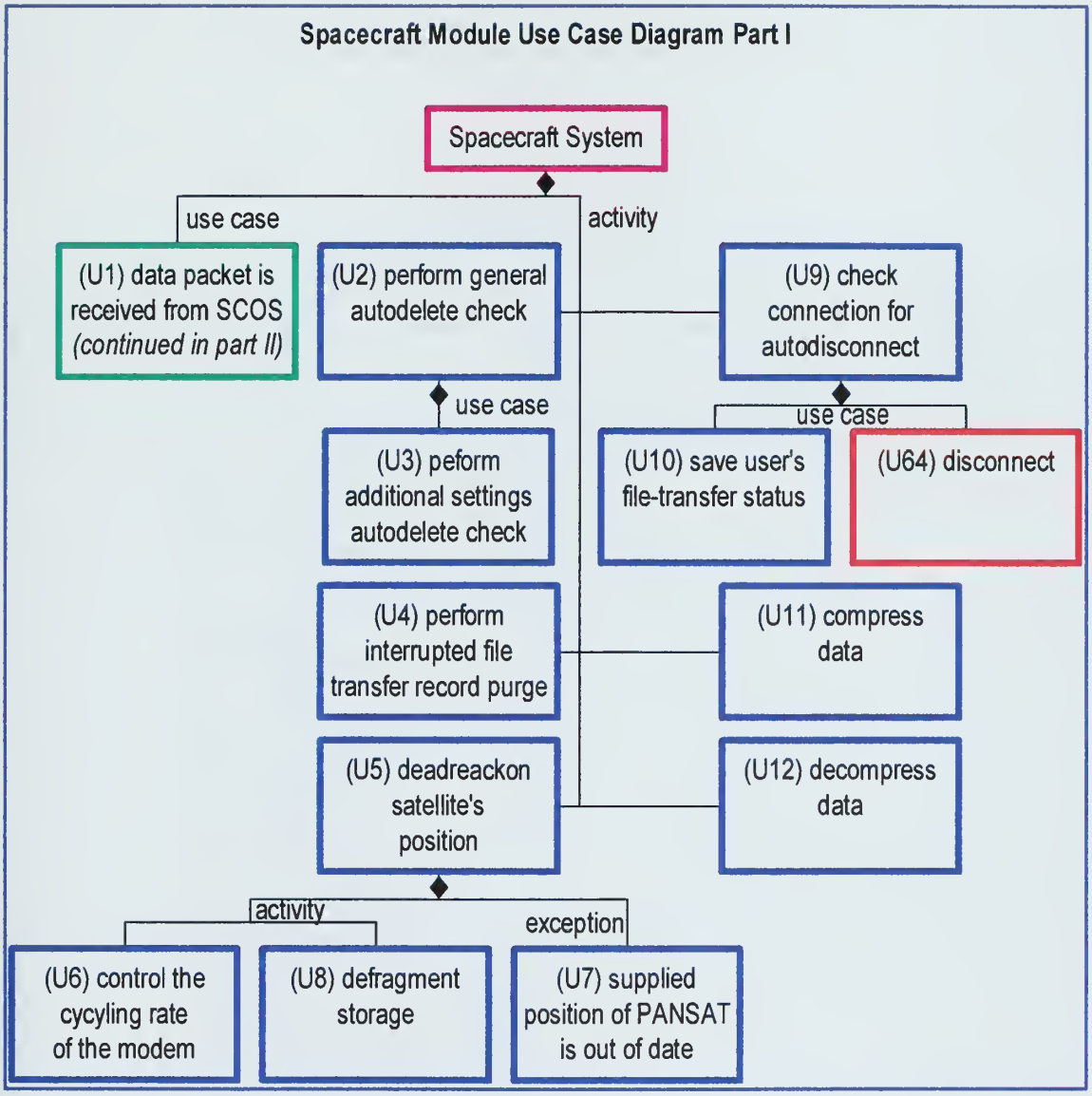
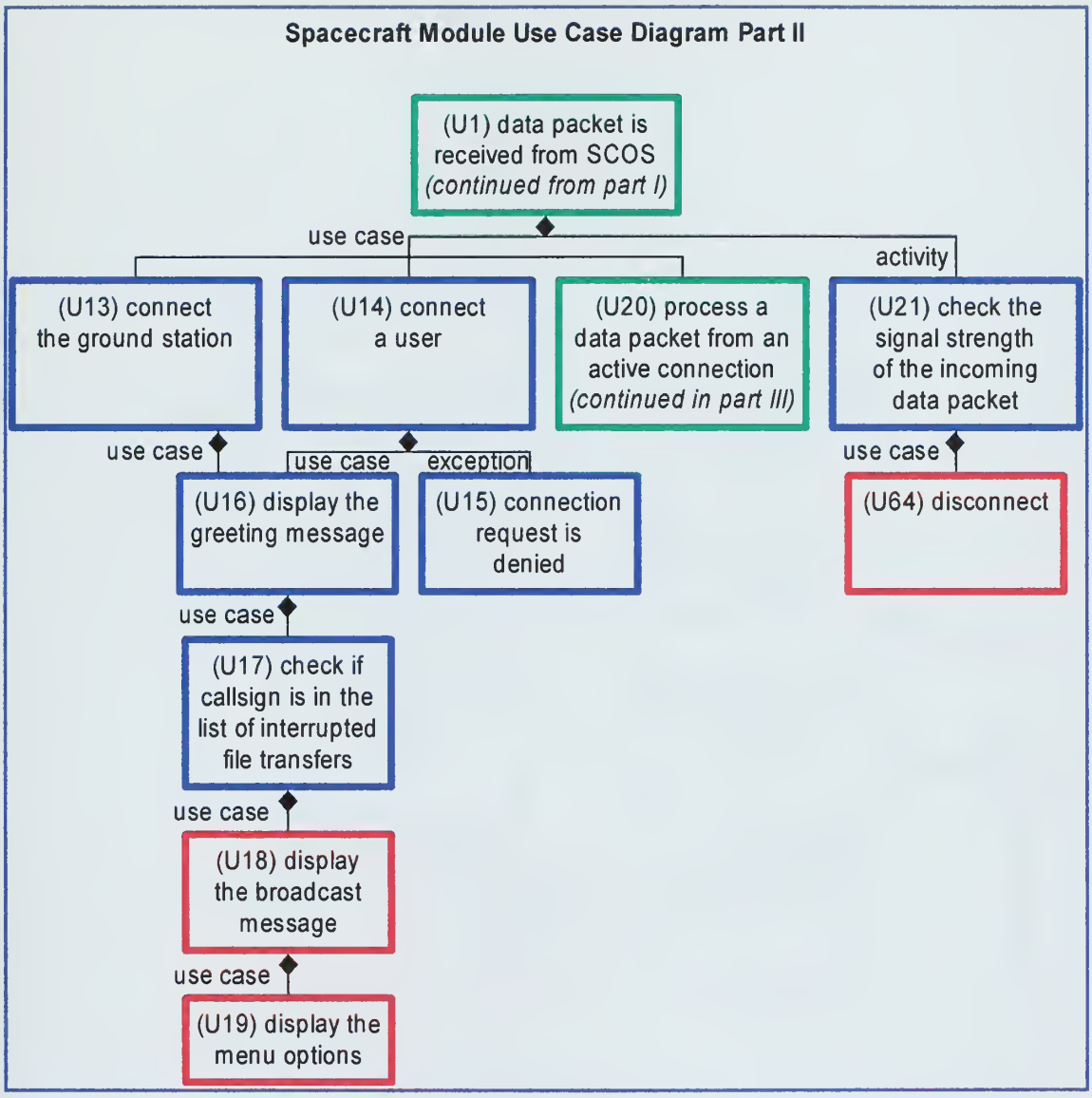
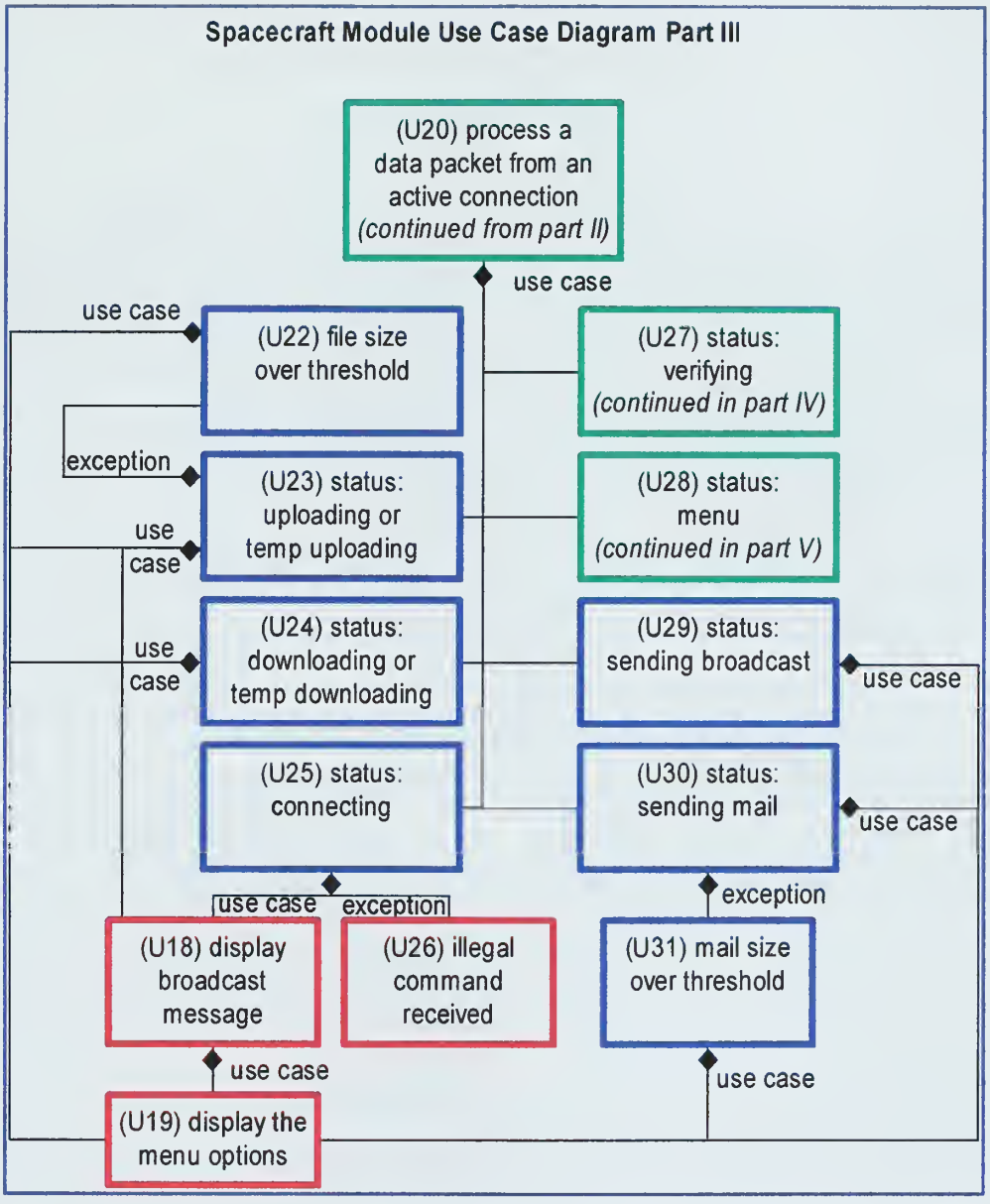
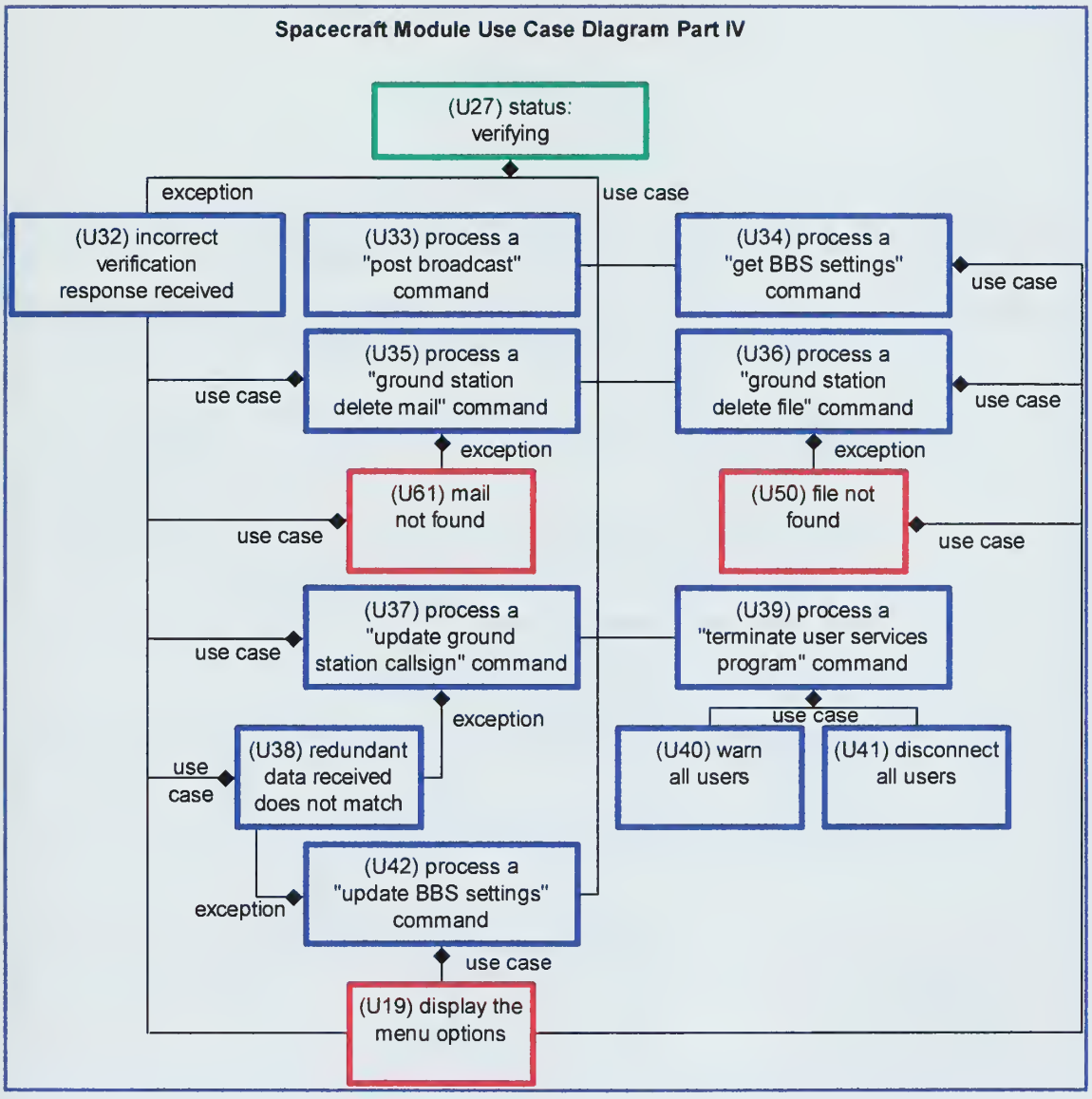
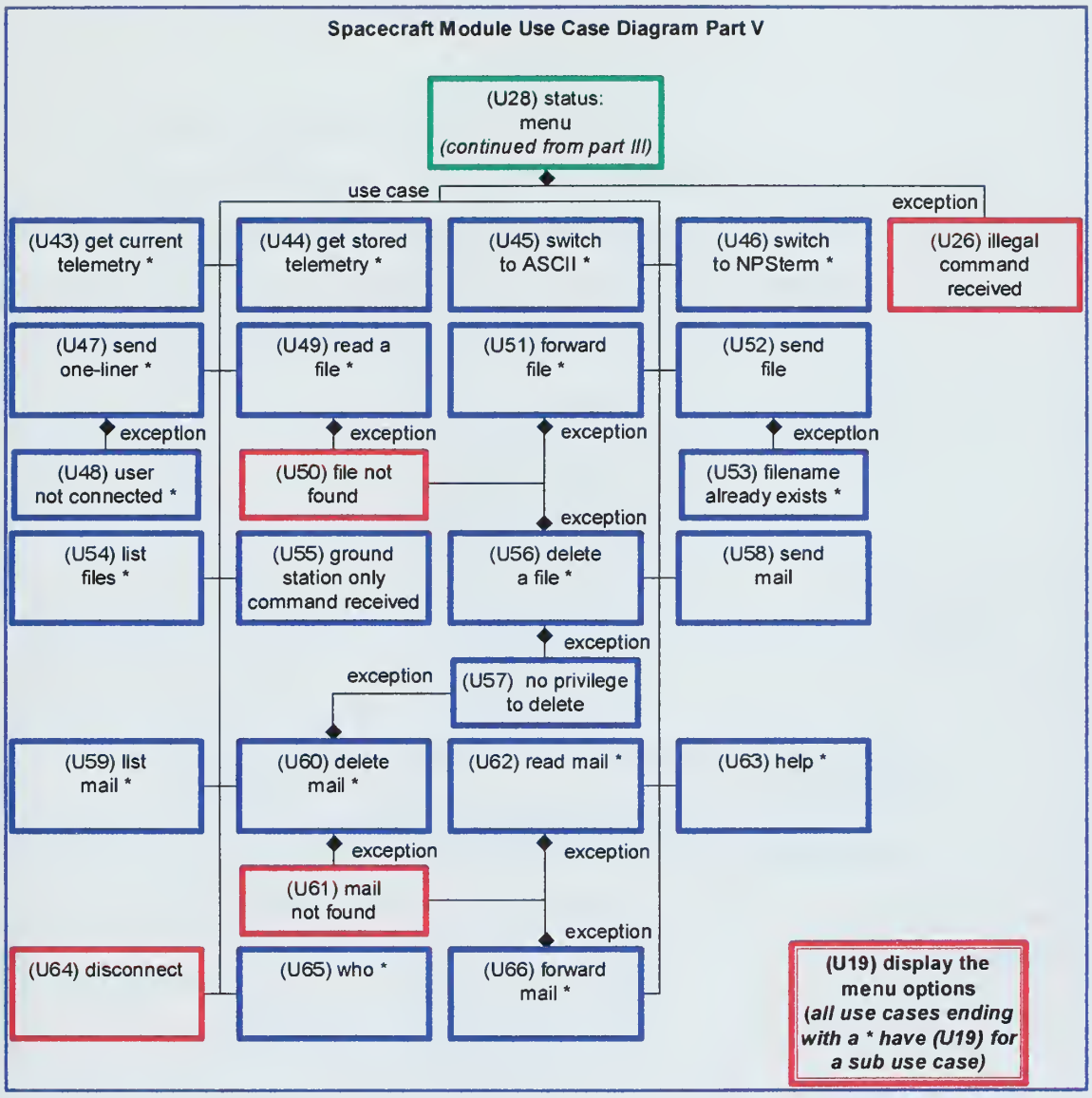


Figure 4 - Spacecraft Module Use Cases (includes Parts I - V)


Spacecraft Module Use Case Diagram Part II
(U1) data packet is received from SCOS
(continued from part I)
use case
activity
(U13) connect the ground station
(U14) connect a user
(U20) process a data packet from an active connection
(continued in part III)
(U21) check the signal strength of the incoming data packet
use case
use case
exception
use case
(U16) display the greeting message
(U15) connection request is denied
(U64) disconnect
use case
(U17) check if callsign is in the list of interrupted file transfers
use case
(U18) display the broadcast message
use case
(U19) display the menu options


Spacecraft Module Use Case Diagram Part III
(U20) process a data packet from an active connection (continued from part II)
use case
use case
(U22) file size over threshold
(U27) status: verifying (continued in part IV)
exception
(U23) status: uploading or temp uploading
(U28) status: menu (continued in part V)
use case
use case
(U24) status: downloading or temp downloading
(U29) status: sending broadcast
use case
(U25) status: connecting
(U30) status: sending mail
use case
use case
exception
exception
(U18) display broadcast message
(U26) illegal command received
(U31) mail size over threshold
use case
use case
(U19) display the menu options


Spacecraft Module Use Case Diagram Part IV
(U27) status: verifying
exception
use case
(U32) incorrect verification response received
(U33) process a "post broadcast" command
(U34) process a "get BBS settings" command
use case
use case
(U35) process a "ground station delete mail" command
(U36) process a "ground station delete file" command
use case
exception
exception
(U61) mail not found
(U50) file not found
use case
use case
(U37) process a "update ground station callsign" command
(U39) process a "terminate user services program" command
use case
exception
use case
use case
(U38) redundant data received does not match
(U40) warn all users
(U41) disconnect all users
exception
(U42) process a "update BBS settings" command
use case
(U19) display the menu options


Spacecraft Module Use Case Diagram Part V
(U28) status: menu (continued from part III)
use case
exception
(U43) get current telemetry *
(U44) get stored telemetry *
(U45) switch to ASCII *
(U46) switch to NPSterm *
(U26) illegal command received
(U47) send one-liner *
(U49) read a file *
(U51) forward file *
(U52) send file
exception
exception
exception
exception
(U48) user not connected *
(U50) file not found
(U53) filename already exists *
exception
(U54) list files *
(U55) ground station only command received
(U56) delete a file *
(U58) send mail
exception
exception
(U57) no privilege to delete
(U59) list mail *
(U60) delete mail *
(U62) read mail *
(U63) help *
exception
exception
(U61) mail not found
exception
(U64) disconnect
(U65) who *
(U66) forward mail *
(U19) display the menu options (all use cases ending with a * have (U19) for a sub use case)

B. SYSTEM CONTEXT MODELS

The System Context Model gives a structural overview of the system's environment. The diagram shows the component being developed relates to the other elements in the entire system [Awad page 43]. The purpose of this model is to give an external perspective of the package being developed as a whole.

1. Spacecraft System Context Model

This figure represents the message flow and content of the messages passing between the components on the satellite, with the User Services module being the center of focus. The other system wrappers are in the brown boxes and the PANSAT User Services system is in the pink box.

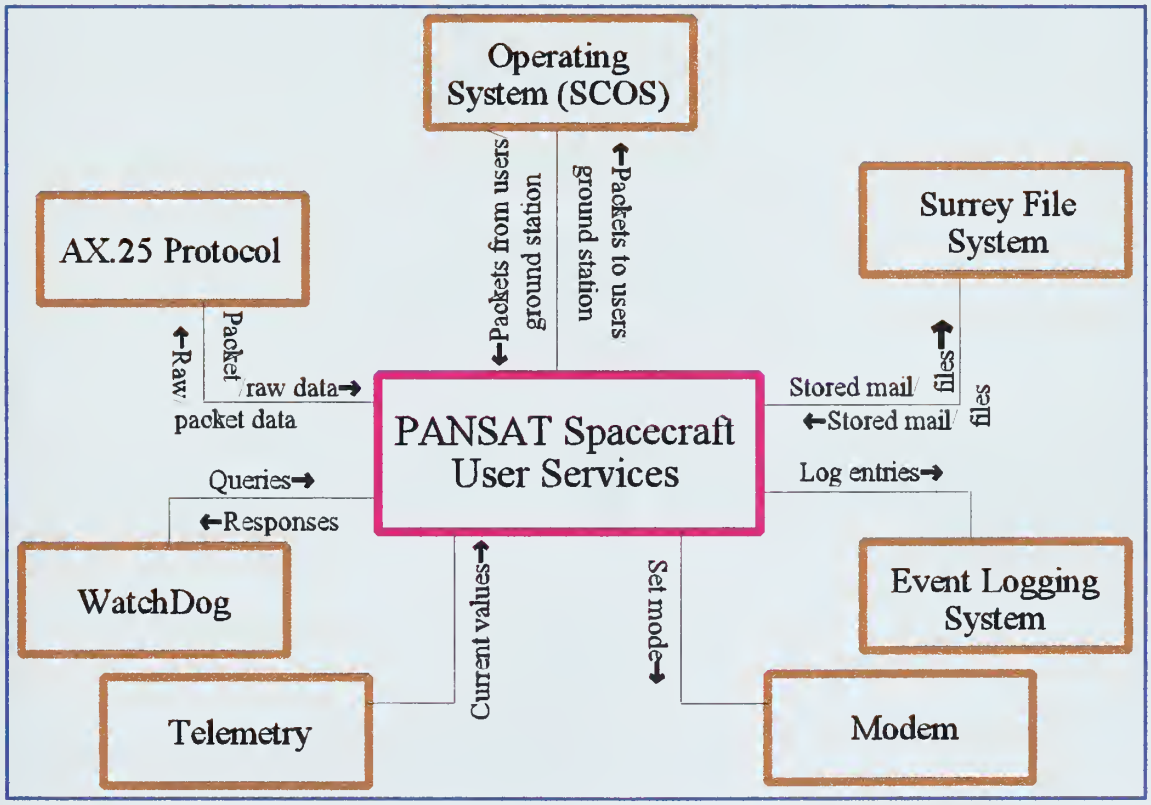


Figure 5 - Spacecraft User Services Context Diagram

2. Ground Station System Context Model

Once again, this figure represents the message flow and content of the messages passing between the components, with the User Services program being the center of focus. However, in this case, the other components are elements of the Windows NT operating system and network. The only component which is an exception to this is labeled with an '*'. The other Windows NT and other wrappers are in the brown boxes and the PANSAT User Services system is in the pink box.

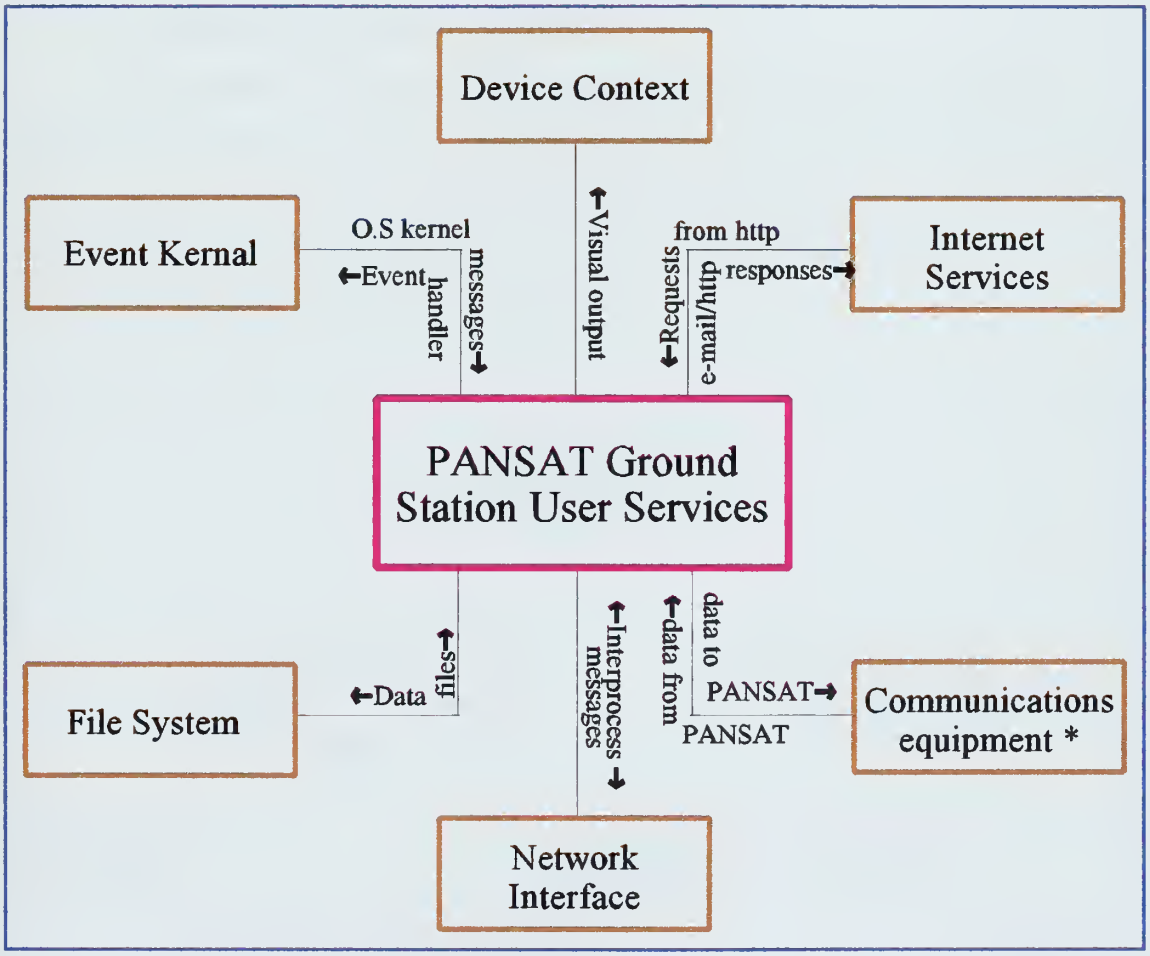


Figure 6 - Ground Station User Services Context Diagram

C. SUBSYSTEM MODELS

Whereas the context model focused on the relation of the developed component to the other components comprising the system, the subsystem model describes the relation of the parts *within* the developed component. A subsystem envelops all the elements of the program which can be defined in a single domain. Elements will *only* make sense in *that* domain and are conceptually partitioned from elements not in the same domain. Subsystems, however do not partition the conceptual space of requirements [Shing].

1. Spacecraft Subsystem Model

On the satellite, all the functionality can be grouped into six major categories. While these groups are not physically separated in the source code, they are distinct. The elements of the PANSAT user services software are in pink, the hardware wrapper is in brown, and the overall system is in black. Dotted and dashed lines are used to differentiate between different dependency cases.

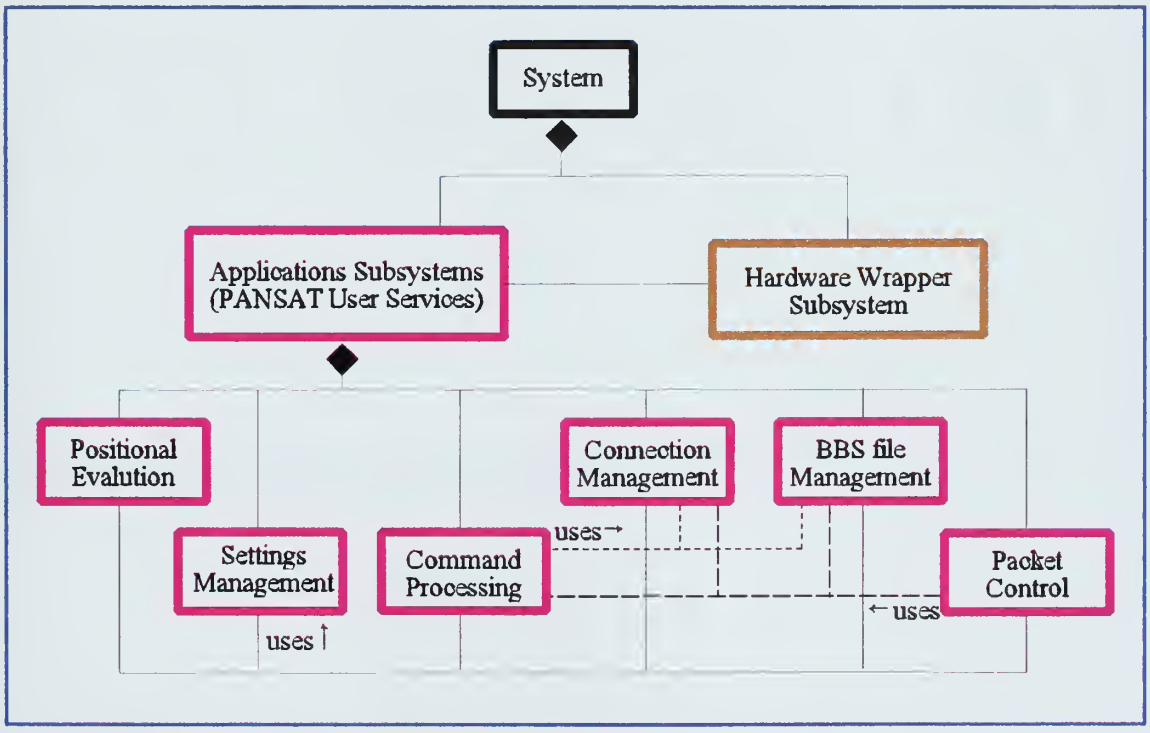


Figure 7 - Spacecraft Module Subsystem Diagram

2. Ground Station Subsystem Model

Unlike the spacecraft module, the ground station's subsystems are not only conceptually separated from each other, by are physically isolated as well. Each subsystem is an independently executing thread, connected with each other only through the message passing depicted in the diagram. Once again the elements of the PANSAT user services software are in pink, the hardware wrapper is in brown (which is mostly the Windows NT operating system), and the overall system is in black. Dotted and dashed lines are used to differentiate between different dependency cases.

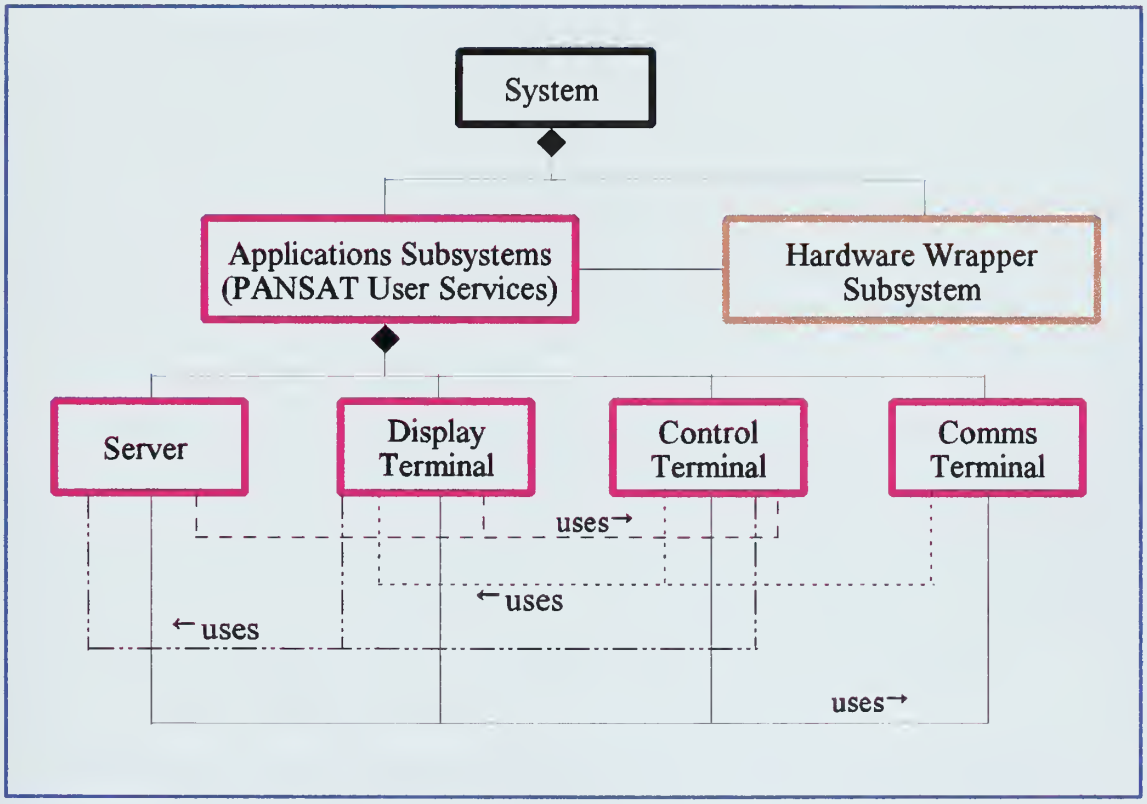


Figure 8 - Ground Station Subsystem Diagram

VI. ENHANCEMENTS TO PANSAT MICRO SATELLITE SYSTEM

This chapter details the initial software experimental features implemented on PANSAT. The three subsystems described have not been incorporated into any existing micro satellite.

A. POSITIONAL AWARENESS

1. Introduction

As previously mentioned, PANSAT has no inherent awareness of its attitude or position over the Earth. Normally, this is not an issue since the antenna configuration is omnidirectional and the satellite's primary mode of operation is to simply respond to data received in a transmission. This passive operational mode is thus event driven rather than position driven. Furthermore, typical internal functions are time-based rather than position-based, meaning they occur on a clock schedule instead of where the satellite is located.

With PANSAT, however, better utilization of resources can be achieved if the spacecraft knows its own approximate position. Power is a premium commodity onboard the satellite. Conserving energy increases the efficiency of the satellite, ensuring power is available at critical times. Additionally, power conservation also extends the lifetime of the batteries by reducing the number of charging cycles, a fixed lifetime number. PANSAT's biggest consumer of power is the communications system (transmitter/receiver). The desire is to temporarily shut off the power on this piece of equipment when the satellite is over areas where there is little chance of communications. Since PANSAT's user base is ground-based HAM radio operators, when the satellite is over water, no communications are likely to take place. Using this premise and factoring in that over two-thirds of the Earth's surface is water, a large energy savings would be reaped if the spacecraft were to reduce power consumption while over the oceans.

A further better utilization of resources would be selective invocation of "housekeeping" functions. Some of the "housekeeping" functions can be processor intensive, yet need not to be performed at a specific time. Examples of this are defragmentation of memory storage or purging old data from the system. By relegating these activities to periods when there are no likely communications, operations will not be impaired by their performance. Also it is preferable to perform the "housekeeping" operations while PANSAT is in sun-light. In sun-light there can be extra solar power that is not used to charge the batteries. Thus once again, positional awareness provides a better utilization of resources.

Since it is impossible for PANSAT to know its position, the solution to this problem is to provide the satellite with a pseudo positional awareness. By furnishing some initial orbital data, the satellite should be able to deadreckon, or predict, where its position should be using the theory of orbital mechanics. This approach appears not to have been previously implemented on any micro satellite.

This pseudo positional approach consists of three steps. First, the ground station needs to provide an initial orbital position and the time of that position. The satellite stores this data and uses it as the basis for position calculations until the data is updated by the ground station. Second, PANSAT performs the estimation of its position. The process of arriving at this estimation is explained in the next section. Lastly, the derived position is compared to a table containing the bounds of the world's oceans. If PANSAT is within these bounds, it can assume that it is over water and, thus, in an area of no likely communication. The first and third step are straight forward. The second step, however, needs to be mathematically derived.

Before developing the algorithm to estimate the satellite's position, two mitigating circumstances need to be considered. The first is that the PANSAT CPU is a 7.4 MHz 80186 without a math coprocessor. Floating point operations must be simulated by the system software. These operations take a relatively large amount of instructions, and correspondingly CPU cycles, to perform. Therefore, to avoid "bogging down" the CPU, a minimum of floating point mathematical operations per unit time is desirable. Balancing

out a desire for no floating point operations with the need for a continuously accurate position, a compromise was estimated that one position evaluation per minute would suffice.

Another element for consideration is that any algorithm implementable onboard the satellite's computer will have its accuracy deteriorate with time. Therefore, periodic updates by the ground station of an initial position are required. The routine onboard PANSAT should be able to accurately maintain itself for a couple weeks, however, in the event that an update is not available for an extended period of time.

2. Background of Orbital Mechanics

The model for developing PANSAT's positional algorithm is Keplerian Theory, which indicates how stellar bodies act under the influence of gravitational forces. This is a simplification of the problem in that only two bodies are considered in this model. In reality, not only do the satellite and the Earth need to be considered, but the other effects need to be factored in as well, including the Sun, the Moon, drag due to atmosphere, and lumpiness of the Earth's gravitational field. In the case of deadreckoning, however, this simplification is satisfactory [Battin pages 191-192]. The orbital accuracy can be maintained by Keplerian theory for several weeks before the other factors become significant. As long as the ground station provides periodic updates, the deadreckoning should not go long enough to transcend the bounds of Keplerian accuracy.

The periodic updates of the satellite's position will be obtained by the ground station via NORAD/NASA's two-liner messages. Every few days, NORAD publishes a complete listing of the positions of over eight thousand orbiting objects. These positions are derived from active tracking of all objects rotating around the Earth, and the application of extremely detailed orbital modeling. The two-liner message provides the satellite positions in terms of Keplerian Elements [Kelso pages 1-3].

Keplerian Elements are a means of precisely identifying an orbit. All the elements are measured in real numbers representing angles. There are six primary elements, although more data is given in NORAD's message. Any data aside from the six primary

elements, however, can be derived from those six elements. Therefore, when a position report is forwarded up to PANSAT, only the six primary elements will be included. The graphic representation of the Keplerian Elements is displayed in Figure 9, which also shows the elements being mapped to a solar system fixed, three dimensional coordinate system (X_I, Y_I, Z_I).

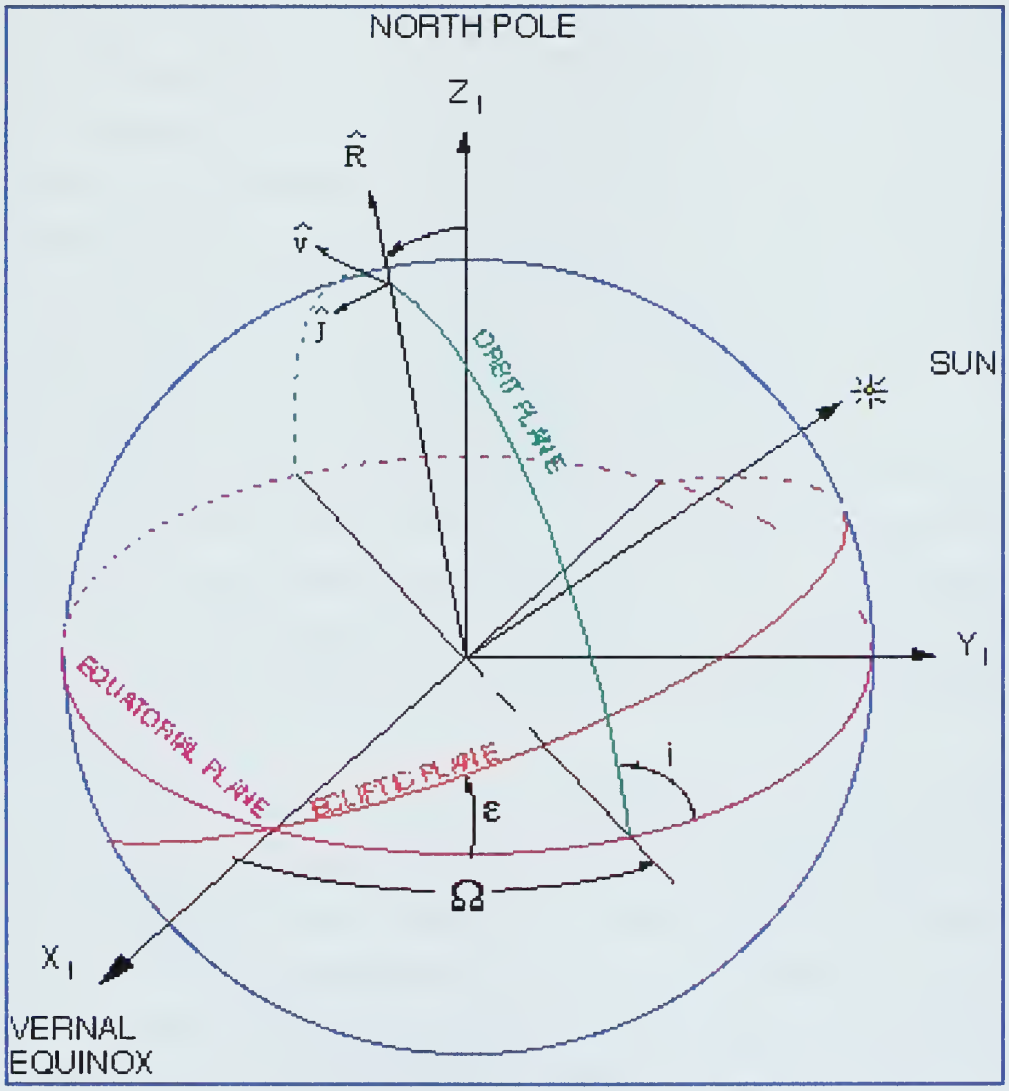


Figure 9 - Orbital Coordinate System and Keplerian Elements

The first element is the inclination (i) or orbital plane. This data indicates the tilt between the orbital plane and the equatorial plane. Secondly, the Right Ascension of

Ascending Node (Ω) indicates the point that the orbital plane crosses the equator going south to north, in degrees from the vernal equinox. Eccentricity describes the elliptical shape of the orbit. An eccentricity of zero means the shape of the orbit is a circle. Next, the Argument of Perigee signals the point on the orbit at which the satellite is closest to Earth. This defines the orientation of the ellipse. Obviously, if the orbit is a circle, this parameter is meaningless. Mean Anomaly tells the degree between the perigee and the current position on the orbit at the epoch. This distinguishes exactly at what point in its orbit the satellite is in, at the time that the position is determined. Finally, the sixth element is Mean Motion, which is the mean angular rotation rate of the satellite in revolutions per day. This translates to the speed of the satellite [Antonio pages 1-4].

While not explicitly listed as a Keplerian element, a seventh element needs to be used with the above positional data. The Epoch, or time of the position, must be supplied for the positional data or the position is meaningless. The Epoch is included in NORAD's two-liner message and must be sent up to the satellite with the other elements.

As depicted in Figure 9, the coordinate system that the Keplerian Elements is translated to is solar system fixed. In the (X_1, Y_1, Z_1) coordinate system displayed, the X axis points to the sun at the instance of the vernal equinox. The Z axis corresponds to the North Pole. The Y axis is then derived from the "right hand rule" applied to the previous two axis. This system is not affected by the rotation of the earth and is constant within the entire solar system.

It is necessary to translate the satellite's non-fixed coordinate system to a fixed system in order to give it a frame of reference. This frame of reference can then be translated into Earth's non-fixed coordinate system, latitude and longitude. Thus the satellite position problem is actually broken into two parts: (1) determine the current position of the satellite along its orbit in terms of the solar system fixed coordinate, and (2) translate the solar system fixed coordinate into Earth coordinates.

The means of determining the first part problem is by applying the following differential equation:

$$F = -\gamma \frac{Mm}{r^3} \tau$$

γ = ratio of inertial to gravitational mass

M = mass of Earth

m = mass of satellite

r = radius from center of Earth to satellite

τ = vector direction of F

Unlike many physical body motion differential equations, not only does numerical integration provide a solution, but an analytical solution can be determined as well [Battin pages 192 - 198].

Much like typical vector problems in physics, such a cannon ball trajectory problem, the future position of satellite is determined using the current position, velocity, and acceleration values of the object. The position and velocity values are provided by the Keplerian Elements. The acceleration is provided by the given equation provided above. Since velocity is first order differential equation of position and acceleration is the second order derivative of position, integration can be use to determine the satellite's next position for any specified given time frame.

There are several methods of numerical integration, ranging from the simple and not very accurate to the complex and highly accurate. Furthermore, as mentioned above, an analytical solution exists which, if used precludes the necessity to use numerical integration. The method to employ onboard PANSAT needs to be determined via experimentation [McGhee].

3. Implementing the Solution

The process of choosing a better solution method is comprised of determining the best numerical integration method and comparing it to the analytical solution. Once again, the constraints on the solution must be that it can maintain orbital tracking in one minute

intervals, for a period of a couple weeks. Additionally, it must use a minimum of floating point operations. While "use a minimum" is a bit vague, the goal was to have six or less operations, but that number was somewhat flexible. The following results were obtained through experimentation.

The Euler method of numeric integration has a linear convergence on the solution. While the number of floating point operations is better than anticipated, its performance was substandard. In practice, it was not even able to keep an accurate track for one revolution, using a one minute integration interval. At intervals of one-half second, the Euler method tracked accurately, but this greatly exceeds the maximum workload of the CPU.

The Heun method of number integration has a quadratic converge. Naturally, the number of floating point operations increases over the Euler method. In fact, the complexity of the algorithm is right at the acceptable level for use on the satellite. The performance of the tracking, notwithstanding, was a vast improvement over Euler, with Heun being able to accurately track the satellite for about a week. Unfortunately, at about the seven day point, the accuracy drops off to an unsatisfactory level. Therefore, Heun integration does not meet with the specifications of a couple of weeks of accurate tracking.

The third method of numerical integration tried is called Runge-Kutta 4. This method has a convergence of the forth power. In fact, the algorithm was able to correctly track the satellite for several weeks, well beyond expectations. Regrettably, however, the complexity of the algorithm was way above the floating point operation limit dictated in the specifications. There, the algorithm was too "expensive" to implement [McGhee].

None of the numerical integration solutions to the position determination problem met with the restrictions incurred by the limited hardware of PANSAT. When implementing the analytical solution, conversely, the tracking results compared with those of the Runge-Kutta 4 method. The analytical algorithm was able to keep accurate track of the satellite at one minute intervals for several weeks. Additionally, the complexity of the

algorithm corresponded to the level of the Heun method, or roughly matched desires for the number of floating point operations.

Clearly, the optimal choice was the analytical solution. The function was implemented by taking the six Keplerian Elements as well as the time since the elements were obtained as parameters. The function then returns the current latitude and longitude corresponding to the point on the Earth's surface the satellite is over. The source code for the positional determination function is listed in Appendix A.

B. FAULT TOLERANCE PLAN

1. Introduction

As described in Chapter I, the PANSAT project is funded by the Navy Space Systems Division (N63). Primarily, the goal of this project is to be an educational tool for military officers as well as a learning experience for the Space Systems Academic Group. However, PANSAT also serves a significant secondary purpose. Currently, the military uses an eclectic, yet expensive, group of satellites for its communications. Many of these satellites are leased, and most of those that are not are quickly approaching their programmed lifetime. Replacing the satellites under the current architecture is expensive. Therefore, the Navy is evaluating the communications capabilities of inexpensive, yet dependable, small satellites with the idea of using them to augment or replace the current system. With this in mind, the PANSAT project is implemented with a minimum of cost, meaning that hardware resources are limited and redundant features are minimal. The inexpensive PANSAT satellite will be evaluated in an established communications environment to see if it can provide the reliable functionality that the military requires. If the experiment proves successful, PANSAT may very well become the cornerstone for a new military satellite communications network.

One of the difficulties with operating satellites, and one that severely impacts its reliability, is that all problems which might arise with the unit must be detected and handled either remotely or by the spacecraft itself. Despite this difficulty, there are means

of obtaining a reliable system, however. Unfortunately, mitigating circumstances in making this particular system dependable are PANSAT's counteractive goals of reliability and low cost. This makes detecting and handling errors even more of a problem to implement. Typically, to ensure a reliable space system, subsystems are implemented redundantly. However, this dramatically raises the costs - a luxury that cannot be afforded in the PANSAT project.

Space itself is a compounding factor for errors occurring. In space radiation is much stronger and thus has much more of an impact on the operations of digital mechanisms. This effect, commonly referred to as "space anomalies," typically reveals itself in the flipping of bits in digital circuits. The side effects of bit flipping are specifically addressed later, but could range from being no effect to crashing the entire system. Another feature of operating in space is that if one element on the satellite fails, it cannot be replaced. However, the entire mission should not be terminated because of a single part failure. Rather the system should be able to respond and adapt to overcome the failure using the resources available to it.

Responding to the two above mentioned space related problems, as well as any others that might arise, is covered under the auspice of *fault tolerance*. Fault tolerance is the means of making a system able to handle errors or problems that might occur during operation. Developing a complete and proper fault tolerant plan should enable the system to gracefully detect and handle every type of problem that might arise during operations. There are many different methods involved with fault tolerance, thus only the areas pertaining to the particular situation of the PANSAT implementation need to be identified [Lee pages 4 - 8].

While some fault tolerant techniques may have been applied to previous micro satellites, none has instituted as comprehensive a plan as designed for PANSAT. Additionally, this is the initial implementation of the Watchdog concept. Watchdog, explained later, was developed for PANSAT in order to conduct system evaluations.

2. System Evaluation

Before a fault tolerant plan can be developed, a survey of the resources available to the system needs to be conducted. PANSAT uses a space-rated Intel 80186 central processing unit (M80C186XL). While this is an older processor, it is space hardened - meaning that it is resistant to space anomalies. Furthermore, the 80186 is inexpensive and fully capable of handling the work load expected of it. The spacecraft has two processors. The second processor merely monitors the first processor. If the first suffers a complete failure - meaning it shuts down due to some other independent power control system turning off the processor - the second processor takes over for the first.

The system has 512 kilobytes of working memory. This is error-checking, volatile random access memory (RAM). There is no redundancy of working memory, however this memory bank uses error-correcting Hamming codes to correct single bit flips. These codes can detect two bit flips, but not correct them. For three of more bit flips, the errors may not be detected. For long term storage, there are two four megabyte memory banks. These memory banks are also volatile RAM, however they do not use an error correction scheme. Initially, these two banks will be set up "mirroring" each other. If data is lost in one bank, the other bank may still contain it. However, if the two banks disagree about the contents of the data, the system will be unable to determine which data bank contains the correct version. The data stored in these memory banks will be the communication data, such as e-mails and binary files.

In addition the four megabyte storage, there are two 512 kilobyte non-volatile flash memory banks. Unlike the other memory banks, if the power is temporarily lost on the satellite, these banks will not lose their contents. Both working memory and the large storage banks will be completely erased even for a minuscule power outage. As the large store banks, these banks will be initially mirroring each other. These memory banks will be used to store telemetry and other important data.

If experiments prove that the mirror memory configuration is not required, the mirroring will be turned off by the ground station. Without mirroring, the memory banks

will put in tandem, doubling the storage memory capacity of the system. The working memory, however, will be unaffected.

The operating software for PANSAT will not be launched with the satellite. Rather, only a small bootstrap program is located in the system's read-only memory (ROM). This bootstrap is set up to establish contact with the ground station, then have the operating system and functional programs uploaded to the satellite. After uploading, the programs are executed. Only then will PANSAT become an operational communications facility.

This uploading software concept has the benefit of being able to upload new versions of the software as required, allowing bug fixes or incorporation of additional features. Unfortunately, if the satellite resets due to a power loss or some unforseen error, all the working programs must be reloaded from the ground station. In fact, if one program is unexpectedly terminated, then that one program needs to be re-uploaded. Termination means purged from working memory and no other copy of a program exists onboard the satellite.

The operating system is a multi-threaded kernel made by the BekTek corporation called SCOS. All the satellite functional programs execute as threads on top of the operating system. The threads can communicate with each other using data streams, which are sent via SCOS. Of course, since only one processor is actually operating, the multi-threaded environment is obtained by interleaving the thread executions.

The main and largest thread running on PANSAT will be the User Services module. This is the program that actually performs the communications interface functions for the satellite. Most of the other threads support the operation of, or interface with, User Services. The scope of the fault tolerance plan defined later is centered around, and in terms of, this main software module.

Some redundancy is obtained in the solar panels, sensors and batteries. Essentially, if one solar panel fails, it will be discounted by the system. If more than a couple of solar panels fail, however, the satellite will not be able to generate enough power for the requirements. Thereafter, PANSAT's performance would be sporadic at

best. If one of the two battery packs fails, the life of the spacecraft is not just halved, it is cut exponentially. This is due to the life cycle of the batteries depending on full charges and discharges. One battery pack would not be able to be fully charged before it is started to be drained, thus is would not get the full cycles necessary for a healthy battery. Finally, failed sensors will simply have to be ignored and not factored into the telemetry analysis [Bible].

These hardware resources listed do provide some fault tolerance, in that most of the subsystems have backups that can take over if the first element fails. However, if an error less than complete failure occurs, the system can not gracefully recover from it. Because that level of error handling is not furnished in hardware, it is necessary to provide that functionality in software, which is the scope of the plan defined herein.

3. Background of Previous Work

Some of the typical methods associated with fault tolerance cannot be used for PANSAT. The limitations in hardware mentioned essentially precludes their use. One primary method used often is called N-version. In this method, several different versions of the same software are made in complete independence and isolation from each other. The results from each are then compared. The answer that has the most matches is assumed to be the correct one. Comparing answers this way is called voting. Voting not only can be used to compare N-versions, but also the exact same software run on different hardware. For instance, running the same program on multiple processors simultaneously could provide different results if one processor was faulty [Kreutzfeld].

PANSAT cannot use N-version or voting simply because the resources are not available to do so. Since one processor is running at a time, voting is pointless. Also N-version is impractical because of the limited amount of working RAM. There is not enough memory to spend it on multiple versions of the same program.

Another commonly used fault tolerant method is called "recovery blocks." This involves conducting "acceptance tests" on the system. At various phases in the program, the entire program state is recorded. If any one of the acceptance tests fails, the

appropriate state is restored, essentially pushing the system time back to that before the error occurred. If the same state is required to be restored multiple times, an alternate error handling routine is executed. The alternate method may be anything from running a substitute method, ignoring the process that generated the error, if possible, or requesting human intervention to correct a procedure [Kreutzfeld].

While there is enough slack in the system to perform the acceptance tests, there is not enough memory nor extra processing time to keep track of multiple program states. We can, and do as described later, use the acceptance tests in conjunction with other methods.

4. Error Classification

Keeping in mind the hardware restrictions incurred with the PANSAT system, the first step in identifying the fault tolerant plan for the satellite is to determine what kinds of problems could occur. It is impossible to identify every possible error that could occur, since the circumstances under which these errors occur is nearly infinite. However, by creating classes of errors, any fault that might arise should fall into one of the classifications and be properly handled, given the assumption that the error classes are correctly designed [Kopetz page 14].

Only after the types of errors have been classified are the tests which identify these errors developed. These tests, the *acceptance tests* mentioned above, are created for each and every subsystem in the program which is being made fault tolerant. The approach is to identify each subsystem and match up all the possible error classes that could occur in the subsystem. Every possible symptom for each error class is then identified and a test to detect these symptoms is developed. The conglomeration of these test are the acceptance tests for that subsystem. This process must be repeated for each subsystem in the program since each has independent behavior and an error must be isolated as much as possible to aid in correction. Acceptance tests are discussed in more detail after completing the description of the error classes [Pradhan pages 677 - 686].

For the PANSAT project, three types of error classes are evident. The first type is those errors that occur outside of the scope of the User Services software. These are denoted as system errors. Essentially this includes any problem with any of the uploaded software, including the operating system kernel, or any system wide problem. System wide problems are those that cannot be identified with any single program, but affect all the software elements. The second type of error class is problems that could occur within the User Services program itself. These are labeled program errors. This includes design as well as incurred errors. The final error class type for the spacecraft is data errors. These errors are not execution errors, like the other two types. Rather data errors are problems that occur with the information utilized or stored by the User Services program.

Table 5 lists all the error classes specified for the satellite, grouped into one of the three types mentioned above. The following paragraphs detail the problems and solutions, if they exist, of each error class.

Table 5 - Software Fault Tolerant Error Classes

System Errors	Program Errors	Data Errors
• Complete system failure • Operating system failure • Operating system livelock • Watchdog program failure • User Services terminates • User Services livelock	• Logic error in module • Executable modification • Program evaluation system failure	• Dump of all files • Corrupt file • System log dump • Data structure integrity violation

While each of these error classes is distinct, the schemes for handling these errors can often overlap. Thus, before each of the individual error classes are addressed, the overall scheme for fault detection needs to be defined. The main method developed for PANSAT is designated the Watchdog system. The Watchdog system overlaps several of the error class areas and, in many cases, can comprise the entire solution to the problem.

The sole function of the Watchdog system is simply to periodically send all of the other operating threads a query. Each one of the threads would then respond with a status

message. If no message is received or a bad status report comes back, Watchdog then takes appropriate action. According to initial estimations, performing the query cycle once every ten minutes should be more than sufficient, such that the system does not get bogged down performing tests, yet still occurs often enough to catch errors as they develop. The check that each program performs in response to the query is tailored for that particular application. The details for the specific action taken for an error are in the discussion of the error classes.

The Watchdog program is unique in its functionality of regulating faulty components in the PANSAT system. Watchdog interacts with the other elements in the system to identify and error the error. On more systems with more available resources, a typical monitoring program would function as an arbitrator in the voting method, previous described. The monitor would evaluate the multiple answers provided by the system and attempt to choose the correct one. An example of this common monitoring implementation is the Space Shuttle. On the Space Shuttle, one computer monitors the results of two banks of two computers. If the two banks agree, the answer is propagated. If the answers disagree, intervention is required and some computers may be shut down [Lee pages 102, 192].

Unfortunately, PANSAT does not have the resource capability to achieve multiple independent answers. Instead, the Watchdog function oversees activity to isolate an error, then eliminate it. Only then can the correct answer be determined. Thus, Watchdog's critical function is to repair the system rather than masking over erroneous results. This is the only way to overcome faulty behavior in a resource limited environment.

5. System Errors

The first error class is one of the most disastrous. A complete system crash means the entire system resets or starts over. This could happen as the result of a power failure, which is hopefully a remote possibility. It is still a possibility, however, in that it could happen as the result of a traumatic experience, such as the satellite being hit by a foreign object. Unfortunately, once the system is reset, all volatile memory is reset as well. This

means that the operating system and working threads, including User Services, are erased from memory. They need to be reloaded from the ground station. The only way that the ground station will become aware that the software needs to reloaded, or even fixed, is by the lack of responsiveness from the satellite. Once normal operations begin, all communications with PANSAT will be done directly with the User Services software, implicitly handled by the operating system. When User Services fails to respond to the ground station, the ground station can make a connection to the BIOS level of the satellite's hardware. The BIOS level connection is restricted to the NPS ground station only. This is the connection mode that the SCOS and User Services modules are uploaded to the satellite. From this level, the ground station can determine that the operating system and threads are no longer on the system and need to be reloaded.

Once the software is reloaded, operations can resume. However, all the e-mail and binary files that were previously sent up to PANSAT would be lost, since they will be located in the volatile storage banks. The telemetry, system settings and log data should not be lost as they are stored in the non-volatile flash memory. Thus, when the new file system program starts up, it can assume the four megabyte storage banks are blank and create a new file structure in that memory block. In the case of the flash memory, however, the file system needs to check these banks for an existing file structure. If a proper file data structure exists, the system should not create a new file structure, but rather incorporate the files that already exist. Once the file system re-accesses these files, the ground station can download them and view them. Inside these telemetry and log files may be an indication of the reason of the failure. While a random event, such as being struck by a foreign object, may not be avoidable, if an internal reason for the failure exists, the ground station may be able to implement a solution to prevent the failure from repeating. Unfortunately, each system crash must be handled on a case-by-case basis and involves active intervention on the part of the ground station.

Two more types of disastrous errors are the operating system failure and operating system livelock error classes. In terms of a fault tolerant plan, both error classes are handled identically. Livelock is the condition where the program is performing an action,

but no real progress is being made. An example of this would be if the program surreptitiously jumps into an infinite loop with no means to exit. The program is working, there is just no means of accessing it. In both error cases, the symptoms will resemble a system crash to the ground station - no connection to User Services will be possible. This is due to all communications with the satellite being directed or channeled by the operating system.

When the ground station enters the BIOS level connection, however, the operator will be able to tell that SCOS and the threaded programs are still active. Since the operating system has been extensively tested and used, such an occurrence generally should not be attributed to a bug in the program. More than likely, an element of the operating system has been altered by a space anomaly. In order to return PANSAT to working order in the quickest time, SCOS and the operating threads should be terminated at the BIOS level, then reloaded from the ground station. Just as in the system crash scenario, the flash memory will still retain all its data. However, in this case, the four megabyte storage banks will also not have lost any data. The file system should, instead of creating a new file structure, use the preexisting structure. Once the system has become operational, there should be no loss of data. The net loss from these errors would be just the operational time lost before the error was found. Once again, however, this error must be handled manually by the ground station. The remaining error classes are less traumatic and have solutions that can be, at least in part, handled by the system itself.

The next error class that might occur would be an error in the Watchdog program itself. As mentioned earlier, the Watchdog program periodically queries the operating threads. In order to determine when it has errors, the User Services also serves as the Watchdog's watchdog. In particular, if the User Services does not get queried for a period of fifteen minutes (a grace period in addition to the standard polling cycle time), a software flag is raised. The Watchdog program has either unexpectedly terminated or jumped into its own livelock condition. In either case, the User Services program continues communications operations as normal. The first error solving action User Services does, however, is to send a stream to the operating system to restart the

Watchdog - which equates to setting the Watchdog's program counter to zero. If fifteen minutes after this stream has been sent to SCOS and no query has been received from the Watchdog program, the User Services logs the event that the Watchdog is not performing. Since Watchdog's performance is not critical to operations, no other programs need to cease. However, during their next connection, the ground station will be notified that Watchdog needs to be terminated, if it is still resident in memory, and reloaded. With the program reloaded, operations should continue as normal. In this case, the system tries to handle the error by restarting the Watchdog program. Only if the error is more complicated than Watchdog can handle does the ground station need to intervene.

Just as the Watchdog program may crash or go into livelock, the User Services program may do so as well. The Watchdog program detects either of these two error classes when the User Services program fails to respond to the query message. If the operating system reports back to Watchdog that the thread it is trying to send the query to does not exist, the User Services in this case, Watchdog determines the program has terminated and is no longer in memory. The Watchdog then logs the error. Next, Watchdog claims the User Services callsign. By doing this, when the ground station tries to connect into the User Services - the communications functionality - they will get Watchdog instead, which will simply provide an error message informing that the User Services needs to be re-uploaded. The benefit of doing this is to lessen the time the ground station takes in finding the error. Otherwise, the ground station would have to fail trying to connect to User Services, make a BIOS level connection, then determine the problem from there. This way provides the error detection up front.

If, however, the operating system believes that the User Services program is still in memory and just not responding to messages, the User Services thread is probably in livelock. The next step is for the Watchdog program to request the operating system to reset User Services, as described above. If after a reset, the User Services module does not respond to the next round of queries, Watchdog logs the fact, then requests termination of User Services from the operating system. This allows the Watchdog to perform the above paragraph's actions just as if the program had already been terminated.

Just as in the Watchdog error class, simple errors may be handled by the system before the ground station is required to intervene to correct a problem.

That completes the error classes in the system errors type group. This fault tolerant plan centers around the User Services module, since it is the cornerstone of PANSAT's operations. However, just like when the User Services module crashes, Watchdog can detect when any of the other threads that might be on the system fail or enter a livelock state. Watchdog handles these errors the same way it handles User Services failing. The exception, however, is that the other threads do not have their own callsigns to connect to. Therefore, the ground station will have to check the system log entries to find an error in one of these other modules. This is not a problem however, since the ground station will be downloading the log record during every pass of the satellite. Since the User Services module should be working perfectly if the error is in another module, getting and reviewing the log entries would be performed normally.

6. Program Errors

The next type of errors that might occur is the program errors. While the system errors dealt with a complete program, or even the entire PANSAT system, failing, program errors are problems that arise within the program itself during execution. These problems however, do not cause the entire program to fail. Rather, a single component of the program fails or produces a wrong result while the rest of the program works as expected. Without fault tolerance, the program would normally continue executing, using the wrong result obtained from the faulty component. Fault tolerant techniques are used so that when a wrong result is determined, it is evident and can be invalidated or corrected. As stated before, the User Services program is the focus of this determination.

The first error class in this type, a logic or programming error, can be found in any program. Ideally, all logic errors would be identified in the testing phase of program development. Realistically, however, some bugs will be missed and not found until actual operational use reveals them. Unfortunately, the system will not be able to tell the difference between this type of error class and the error of program modification by space

anomaly. Program modifications have the consequence of a module that was once providing correct results now providing faulty results. If the program modification was so severe that it locks the whole program or livelocks when the procedure is invoked, the error class is no longer within the program error types, but becomes one of the system error types. Although both the logic error and program modification error have different causes, the system only sees the same result - incorrect results.

These wrong results are generally found in response to a Watchdog query message. When the User Services program receives a query message, before it replies, it conducts a series of tests to ensure that its subsystems are preforming normally. These tests are the acceptance tests mentioned previously and defined later.

Once a bad subsystem has been identified by the acceptance tests, User Services notifies the Watchdog program, which logs the problem. The ground station can then review the problem and upload the solution. Of course, uploading the fix most likely involves gracefully terminating the User Services program and sending up a fixed version of the code. By performing the graceful termination, the ground station will be able to preserve all the data on the system. Thus none of the communications files will be lost, minimizing the impact of the error. If the program errors caused incorrect data to be stored, however, some of the data may be permanently lost. Meanwhile, if the problem was just a program modification, the code would not need to be modified from the original form. A logic error would require an updated version of the software to be created. The ground station is responsible for determining which of the two error classes caused the problem. This determination will be made by viewing the log data and comparing the results with backups of the satellite software maintained at the ground station.

While the solution to the error is being handled by the ground station, the satellite may be able to keep performing, albeit in a diminished mode. The User Services software will maintain a table of subsystems which correspond to particular services provided by the program. When a subsystem is determined to be faulty, User Services will simply not use or offer the corresponding service obtained from the table. If the denied service is mission critical, such as ability to put data into a tranmissionable packet, the program will

terminate. If the service is not mission critical, it will notify users who log into the satellite that a particular feature is temporarily unavailable, for instance, retrieving the current telemetry might be removed from the menu list if the telemetry subsystem failed. The goal of handling this error is to provide as much communications functionality as possible until the problem is rectified.

A special case of the program modification error is the program evaluation subsystem of User Services itself suffering from a space anomaly. The Watchdog program checks the response sent by the User Services program. If all of a sudden User Services reports errors in every single one of its subsystems, the error most likely lies inside the subsystem conducting the testing, not the other parts. In this case, Watchdog logs the suspected error and keeps operations as usual. The Watchdog program will continue to query User Services, just to determine if an unexpected termination or livelock occurs. The actual response sent back the User Services will be ignored, since the testing system would be expected to be flawed.

In all three of the error classes in the program error type, the ground station is required to intervene to correct the problem. This is due to the programs being held in the volatile working RAM. No copy of the code exists on the satellite to determine which bits might have been flipped. However the key to this fault tolerant technique is that service interruption is kept to a minimum. Early error detection and notification of the ground station, coupled with temporary work-arounds by the system software itself, allows continued services to be provided to the end user until those fixes can be implemented.

7. Data Errors

The final of the three error class types is data errors. For the most part, the previous errors described were dealing with the executable part of the program being incorrect or altered. Data errors, on the other hand, involve the programs working perfectly. The data that is being worked on, however, has been damaged in some way. This means that most of these errors are detected via the acceptance tests.

The data error with the largest scope is the event of all the files in the storage banks being dumped or erased. When this occurs, and the User Services program is still executing, a system reset is not the cause of the storage dump. In this special case, the contents of memory may still be intact, only the file allocation table (FAT) may have been corrupted. The system will first attempt to reconstruct the FAT from what information can be found in the memory location where the FAT is normally kept. Aiding in the chances of rebuilding the FAT is the fact that the FAT is created in duplicate. Thus, out of two corrupted FAT's, the possibility exists for a complete and correct table to be created. Also helping with the recovery of the FAT is the fact the files are created by User Services in a very methodical way. Only two groups of files exist in mass storage, e-mail files and binary files. They all use the name scheme of m#### and f####, where # represents a digit from 0 to 9. Conducting a search through the memory banks may reveal the files are still present and reconstructable. Knowing this and the data structure format of the FAT, there may be enough information obtained to create new FATs from the files left in memory.

If attempting to recreate the FAT fails, the files are unrecoverable. PANSAT will have to establish a new file structure in the four megabyte storage area and start from scratch. For one week after such an incident, any user who communicates with the satellite will receive a warning message that all data was lost on the date that it occurred. That way a user can re-upload an e-mail or file as necessary. In either case, the problem is logged so the ground station can review it. While the ground station cannot restore the files, if the cause of the problem was more than an isolated incident, the ground station may be able to determine the error source and fix it.

A less severe data error is the corruption or loss of a single file in storage. Hopefully, since the storage area is using mirrored files, one of the two storage banks contains a correct version of the file. However, corruption of a file can still occur in both memory banks. Typically, a corrupted file only has a section of the file lost. When a file is determined to be corrupted, User Services will rebuild the file with all the information that could be recovered. A marker will be put in the place where the data was lost. For

instance, in an e-mail message, the place with the missing text would be replaced with the "[text missing]" caveat. Additionally, the originator of the message will be sent an administrative message indicating which message was damaged and indicating a retransmission may be necessary. Of course, the originator would get this warning in the next communication with PANSAT. If no discernable data is retrievable from the corrupted file, specifically the originator of the file, the file is simply deleted from the system. Just as with the file system dump error above, the ground station is notified of the error for evaluation, but will be unable to do anything more in the recovery of that particular corrupted file.

While the Watchdog program would determine a problem in the program that managed the logging system, it would be unable to know if the log file itself had unexpectedly been erased. This is the next type of data error. Since the log file will be maintained in the non-volatile flash memory, erasing it would most likely be the result of an inappropriate software action rather than a hardware glitch. When the system logging program responds to the query from the Watchdog program, it checks to ensure the contents of the log file are intact. If not, the Watchdog program is notified. Unfortunately, with the logging system temporarily non-functional, the Watchdog system is unable to use its primary method of notifying the ground station of errors. In this special case, the Watchdog program sends the User Services a special message informing it that the logging system has an error. In its next communication with PANSAT, the User Services will notify the ground station of the logging problem.

Note that this procedure is used not only for a problem in the log file, but if the system logging program has a fault as well. The ground station will have to determine the nature of the error. If the logging program is faulty, it will be reloaded. If the log file is accidentally deleted, the ground station must determine which program deleted the log file. This will have to be done via a recreation of the satellite's environment at the ground station. As much information about the current state of the satellite will have to be provided by User Services in order to aid in the recreation process. These errors are a special case in that the normal means of error notification, and thus error rectifying, is

removed from the system. Thus, even though operations continue for the satellite, the ability to handle and fix any other error that might happen is drastically reduced until the logging system error is fixed.

The final error class identified for the PANSAT project is the corruption of a program data structure used within the User Services program. The User Services program functions as an automata state machine, acting on data from an end user based on what state the system is in for that particular user. Thus, if a data structure becomes corrupted, a user connection may act unpredictably. Fortunately, the corrupting of a data structure will most likely affect only a single user - the other connections will be unaffected. Once a data structure is determined to be corrupt, there is a good chance that it can be rebuilt. The operating system provides many of the state information as it delivers a packet of data received from the user to the User Services program. Thus, using that information, much of data maintained about the connection can be determined. If, however, a complete data structure cannot be rebuilt from the data provided, the User Services needs to interrupt the connection by sending the user a series of "interrogative" messages. After these messages are sent, the program should put the connection back into the default state - print out the main menu of choices for the user to pick. The user would then be responsible for salvaging the session and retrying to complete the work that was ongoing at the time of the fault.

As with all other data errors, the corrupt data structure error will be logged and as much data will be recovered as possible. The ground station will be notified of the error, but will not be able to fix that particular instance. Using the data, however, a solution to the problem that created the data loss in the first place may be determined, hopefully eliminating future errors of the same kind.

This concludes the classification and handling of errors that PANSAT might see. Although each specific error that might occur is not detailed in this plan, most likely any error that arises will fall into one of the error classes and be handled appropriately. No matter what, PANSAT always has the BIOS level connection which can be used to reset the system to start over. If for some reason this connection is unable to be made, the

satellite has suffered a major catastrophe and, for all practical purposes, is dead. While there can be no mechanism to handle this type of an error, the likelihood of this happening is extremely small.

8. Acceptance Tests

When discussing the program and data errors above, it was taken for granted that these errors would be detected by the acceptance tests mentioned. As mentioned previously, without a complete idea of the errors to create the tests for, the acceptance tests can not be fully designed. Once the error class plan is formulated, however, it is possible to develop the precise tests to fulfill the plan . As implied above, the acceptance tests should test all of the subsystems and data organizations to determine their reliability and integrity [Pradhan pages 677 - 686]. Table 6 shows the four types of acceptance tests the PANSAT User Services program will perform to conduct a self-evaluation. Each one of the classifications is discussed in the following the table.

Table 6 - Fault Tolerant Acceptance Test Classifications

Acceptance Test Classifications
• Satisfaction of requirements
• Accountability tests
• Reasonableness tests
• Computer run-time checks

The first acceptance test type is satisfaction of requirements. This means the subsystem provides the expected results. Naturally, each subsystem will have a different method for checking its results. A standard means for procedures that take data in, process it, and return an answer is to use table comparison. Obviously, it is impossible and impractical to maintain a list of all legitimate answers for any given input. Rather, when the Watchdog program's query message is received, the User Services testing

procedure refers to a table of predefined tests for each subsystem. There should be only one or two tests per subsystem or the program could get bogged down performing all the tests. The tests, however, should be a good representation of the type of calculations required by the system. Included in the table is the correct answer the subsystem should return for the test. Any variance between the table answer and the subsystem provided result would be a logic or program modification error, as described above, and reported to the Watchdog system.

This type of comparison test is used on the subsystem which performs the "packetizing" of data. All data sent from the satellite must be in AX.25 packet format. By providing preselected raw data to the procedure that forms these packets, then comparing the format of the packet to what is expected, a determination of a flaw arising in the procedure is be achieved. Additionally, the comparison test is performed on the data compression/decompression algorithm. A set of compressed data is sent to the procedure to see if the correct data is extracted, and vice-a-versa. Finally, the position determination routine is tested with this technique. The routine takes a initial position and a change in time since the position and produces the current position. The comparison test table contains all three elements to determine the position's routines accuracy.

If the subsystem is not answer based, but rather performance based, the User Services could check that the action performed was actually the expected result. While the answer based checking would be done only in response of a Watchdog query message, the performance based procedures would be checked after the completion of every action.

On PANSAT, the system that saves a file into the storage memory banks is be checked by determining whether the file name and size of the newly created object is the same as that provided to the file system. Once again, a mismatch would be classified as either logic error or program modification error. Additionally, when an entry is sent to the event logging subsystem, the event log file is checked to ensure the entry was appended to the end of the file. The defragmentation module also uses a performance based satisfaction test. After a file has been defragmented, the file data is looked up in the FAT to ensure that it is now taking up on contiguous block in storage.

The second type of acceptance tests are accounting tests. That is, User Services keeps a tally of certain information about the rest of the system. When a particular subsystem is queried about its tally, if it does not match User Services's version, an error is declared.

Accounting tests are used to compare a file counter held by User Services with a directory count provided by the file system. A variation in the number could indicate a corrupted FAT, which is handled as a dump of the file system error. In addition to that procedure, the number of packets returned from the packetizing system for a certain sized block of data should be a predetermined amount. A variation would be a logic or program modification error. Next, a timestamp is placed on all recorded activities of the User Services program, such as saved files, transmitted packets, or positional estimates. If the timestamps are not ordered, the system's clock interface system may be flawed. Finally, connection records are checked to ensure that the number of in use records matches a counter which indicates the number of ongoing connections. If the numbers differ, a connection has been lost. The handler for a data structure integrity violation is invoked.

The next acceptance test type is called reasonable tests. This name says it all - whether the system provides a reasonable result. This is the most common acceptance test and is used with almost every subsystem in the User Services package.

For instance, this test is used when telemetry values are obtained. Each value procured has a possible value range located in a table. If the value is outside of the range, or the value has changed within the range, but at a rate that is not physically possible, then the means of getting the values or the sensor providing the value may be faulty. Manifestly, if an end user requests an operation that "does not make sense," the operation should be rejected and a warning sent back to the user. If data returned from a function is not the correct size, it is faulty. This includes the size of packet received from SCOS to the length of a datastream connecting processes. A position returned by the deadreckoning procedure must be within a range of possible positions on the satellite's orbit. A position of the North Pole would be flagged as faulty since PANSAT's orbit does not come close to the North Pole. Finally, settings commands received from the ground

station must make "sense." That is, the settings must correspond to feasible operations. For example, performing a auto-purge of the storage banks every minute is not a practicable operation.

The last type of acceptance tests are labeled computer run-time checks. These are handling typical run-time errors that usually cause a program to terminate. Also called exception handling, these functions catch divide by zero, overflow, underflow, and other errors by abstracting their operation into a single safe procedure. By trapping their execution, the program will continue to execute, and the subsystem that requested the illegal operation can be determined. This is implemented by placing all the exception possible routines into a single checked module. Once the module is proven safe from crashing, any element that uses the module is safe from crashing due to checked run-time error. It is important to keep the program from terminating since, as mentioned before, once the program terminates, it is removed from memory. Removal would prohibit determination of the offending subsystem thus exacerbating the fixing of the error.

Another run-time check is for each procedure to check the program stack upon being called. If the stack is not formatted correctly, the procedure may have been illegally jumped to. A correct stack format would be indicated by the proper type and number of parameters being placed on the stack. It may be impossible to determine where to resume execution from when returning from an improperly called function, so a safety point of reference, the main menu in this case, may have to be jumped to in order to continue execution.

These acceptance tests are very specific to the individual subsystem that they are evaluating or protecting. The basic concepts presented here form a comprehensive coverage for many systems, however. When developing a list of tests, every subsystem should be evaluated. In general this type of fault tolerance plan increases the overall work load by 25 percent, an acceptable amount.

9. Conclusions

None of the technologies applied in this fault tolerance plan are difficult in and of themselves. However their application in conjunction provides integrated comprehensive coverage, which dramatically increases the reliability of PANSAT. As mentioned at the beginning, most fault tolerant research has been working with distributed systems, working with a large pool of hardware resources [Kreutzfeld]. Unfortunately, the particulars of space, especially in this project, preclude the use of most of those features. The methods instituted on PANSAT are tried and true fault tolerant methods, only they have not been previously linked together to form a consolidated fault protection for a system. Using this innovative plan, the satellite should be able to operate and provide a level of reliability expected by the military for a communications satellite.

Limited resource fault tolerance can be successfully implemented. While it may not provide the same level of *self*-correction that is expected of a large resource system, the errors are still trapped and handled by the limited system - with a little bit more of an active role by the ground station.

C. OPTIMIZED PROTOCOL

1. NPSterm Introduction

Due to the nature of its orbit, PANSAT has a limited window of opportunity for communication with a user during a pass over the user's facility. Typically, communications can be maintained with the satellite for periods of two to nine minutes, depending on the particular orbital pass. The average length of the window of opportunity time is approximately six minutes. Since time is at a premium and the bulk of operational time is consumed transmitting data, PANSAT implements a new application-layer protocol to optimize communications. This new protocol, designated NPSterm, reduces the size of the information required to be transmitted to and from the satellite. Fewer bytes of data required to be transmitted means less time consumed by the transmission.

Thus, in the same amount of time, NPSterm allows more operations to be performed than when an ASCII interface is used.

The standard application protocol utilized by the HAM community is a simple ASCII interface. Characters are transmitted exactly as they are typed by a user. While this has the ability to be universally understood by almost any terminal, it is a wasteful protocol in that the information is contained in a comparatively large number of bits.

Even though PANSAT maintains the ability to communicate using this ASCII interface, ensuring capability with other HAM communications facilities, NPSterm is implemented and is the preferred protocol. To employ NPSterm, the end user must download PANSAT specific terminal software. This software will be available via the Internet from the PANSAT Ground Station Home Page (http://131.120.25.124/). All connections to the satellite must be started in ASCII mode, but will then shift to NPSterm via a command sent from the terminal software. The use of the protocol and switching of protocols is transparent to the end user. The software handles these operations automatically. The only difference an operator can discern is a dramatic increase in data throughput.

NPSterm incorporates two methods to optimize the information flow. First, instead of a series of characters representing the user's command to the satellite, a single bytecode contains the entire command (sans parameters). Secondly, the data in a packet to be sent to or from the satellite is compressed via a zero loss algorithm. NPSterm uses the LZSS algorithm (Nelson). This relatively straight forward algorithm provides an LZ77 style compression. Using this algorithm, the size of normal text data is compressed 58.83 percent. Graphics are reduced an average of 43.45 percent and binary files shrink 41.44 percent (Nelson p. 512). Combined with the command encoding bytecode method, an average data packet using NPSterm will be reduced in size by half.

Although it is beyond the scope of this thesis, a third optimization method can be integrated into NPSterm. Typically seen in modern microprocessor structures, pipelining can be designed into NPSterm. Although pipelining would not cut down on the transmission time of the data, it would allow streamlined processing of commands by the

satellite. Although time processing a single command would remain the same, throughput for a series of commands would increase. Implementing this, however, would take extensive command processing experimentation and evaluation, an undertaking for a follow-on project.

The details used in implementing each of the two methods follow.

2. Bytecode Commands

The concept behind this method is to reduce the number of bytes to represent a specific command to PANSAT to one. In ASCII mode, a command, not including the parameters, can take up from one to eight bytes. Each of these ASCII commands are unique and can be mapped with a one-to-one correspondence to a single byte value. Note that some of the bytes that are saved are due to whitespace delimiters required in the ASCII mode.

Additional savings of up to two bytes by binary encoding numbers in the parameters rather than sending the numbers up in their ASCII format. Whether or not the savings occurs is due to the number of digits used in the ASCII encoding of the number. In a very small number of cases, one in a thousand, the number encoding actually takes up an extra byte from the number encoding. This is due to binary encoding using a static length of two bytes. This may simply negate the gain from removing whitespace. In 99.9 percent of the cases, however, no loss of data size occurs.

Table 7 contains all the possible commands to send to PANSAT. The commands are derived from all text permutations described in Tables 1 and 2. Once the satellite receives one of the bytecodes, the system translates the number back into its corresponding ASCII command, then processes the command as usual.

Table 7 - NPSterm Command Bytecodes (# represents a 16-bit integer, *data* means any number of unchanged text parameters)

Command Title	ASCII Representation	Bytecode	Minimum Bytes saved
Get current telemetry	TC	1	1
Get stored telemetry	TS	2	1
Send mail	SM *data*	3*data*	2
Send file	SF *data*	4*data*	2
Read mail	RM # RM E	5# 6	2 3
Read file	RF #	7#	2
Delete mail	DM E DM # DM # - DM - # DM # - #	8 9# 10# 11# 12##	3 2 4 4 6
Delete file	DF # DF # - DF - # DF # - #	13# 14# 15# 16##	2 4 4 6
List mail	LM N LM U LM E LM A LM U # LM U # - LM U - # LM U # - # LM E # LM E # - LM E - # LM E # - # LM A # LM A # - LM A - # LM A # - #	17 18 19 20 21# 22# 23# 24## 25# 26# 27# 28## 29# 30# 31# 32##	3 3 3 3 4 6 6 7 4 6 6 7 4 6 6 7

Command Title	ASCII Representation	Bytecode	Minimum Bytes saved
List file	LF N	33	3
	LF U	34	3
	LF E	35	3
	LF A	36	3
	LF U #	37#	4
	LF U # -	38#	6
	LF U - #	39#	6
	LF U # - #	40##	7
	LF E #	41#	4
	LF E # -	42#	6
	LF E - #	43#	6
	LF E # - #	44##	7
	LF A #	45#	4
	LF A # -	46#	6
	LF A - #	47#	6
	LF A # - #	48##	7
Forward mail	FM *data* #	49#*data*	3
Forward file	FF *data* #	50#*data*	3
Switch to ASCII	NA	51	1
Who	W	52	0
Send one-liner	M *data*	53*data*	1
Disconnect	X	54	0
Post broadcast message	P *data*	55*data*	1
Get BBS settings	G	56	0
Update BBS settings	U *data*	57*data*	1
Update ground station callsign	GS *data*	58*data*	2
Terminate User Services	KI	59	1

3. Compression Algorithm

For NPSterm, the LZSS compression routine was selected because of its high compression rate and relative code simplicity. Additionally, the operating parameters of a 12 bit index size and 4 bit length were chosen to keep the execution time minimal. Using larger parameters would result in an exponentially longer time required to navigate and

maintain the data dictionary. Furthermore, using these parameters keeps the algorithm's support data structures from consuming an exceptionally large block of memory.

A drawback of LZSS is that when the User Services program first starts up, the compression routine will operate slower than normal. After several uses of the routine, however, the data dictionary will have built up enough to perform quicker compression. Expansion performs extremely efficient all the time, however. A second drawback of LZSS is that it works better on small files rather than large ones. This has small impact, however, since most of the files anticipated to be sent to PANSAT will probably be only a few kilobytes in size.

A shortcut of using the NPSterm is that data received from an end user will already be compressed. Thus, no operations are needed to save the data compressed. It is simply copied compressed from packet straight to mass storage. If a compressed file is requested for download by a user not using NPSterm, expansion of the data is performed onboard the satellite. Because the algorithm needs a minimal of data structures and operations to expand data, it is considered a low cost operation. Thus, these steps save CPU cycles onboard the satellite.

The original LZSS routine, taken from Nelson's The Data Compression Book was designed to work only on files. In order to work in the serial data communications environment of PANSAT, the code has been modified to operate on generic data streams. This abstraction allows operations on both files and packets of data held in memory.

The source code for the modified LZSS routine is located in Appendix A.

VII. IMPLEMENTATION ISSUES

A. REAL TIME TESTING

The ground station normally has relatively little exorbitant processing activity ongoing during the times when communication is not possible with PANSAT. Since the satellite is only within line-of-site of NPS approximately five percent of the time, 95 percent of the ground station's time will be spent relatively idle. However, during that five percent when communications are occurring with the satellite, it is critical that the ground station can process all the incoming and outgoing data as well as perform all of its own processing.

The goal is to ensure that during the periods of communications, all processing can be accomplished in essentially real time. What real time translates to in this instance is the operator at the ground station not being able to notice a delay in processing, other than the inherent transmission delay incurred when communicating with satellites. An example of this is when the operator types in the command to get current satellite telemetry values. As soon as the command is entered, it is immediately sent to PANSAT and, as soon as the reply is received, it is immediately displayed on both the display (in graphic format) and control terminals.

To verify that the network configuration described in early chapters can handle this workload, a series of timing tests were performed on the network. The tests simulated data being received by the communication terminal. The generated data is then disseminated to the other ground station terminals via the 10Mbps Ethernet network. The other terminals then simulate processing the data by repeating all the data back to the communications terminal. The distributed system of the ground station ought to be able to manage all of the processing at a data rate of approximately two times the maximum data rate possible receivable from PANSAT (to allow slack in the system).

For a matter of comparison, not only is a test data packet size of 256 bytes used, which is the maximum packet size expected to be utilized with PANSAT, but also packets of two, three, and four times this size are used as well. Furthermore, the same packet

passing tests are performed on a single PC, with separate threads representing the individual terminals of the ground station. This displays the response rates if the functionality from all four terminals comprising the ground station were implemented on a single PC. These results could be useful if reconfiguring the network is required. Reconfiguration would be necessary, however, only if the rate of processing the data on the network's current configuration does not meet expectations. Although used as a reference point, a single PC implementation is not an ideal operating environment for the ground station management. Finally, the test results include the maximum data rate expected from PANSAT. To put perspective on the results, the maximum data rate of a T1 line is shown as well. The results of the experiment are displayed in Figure 10.

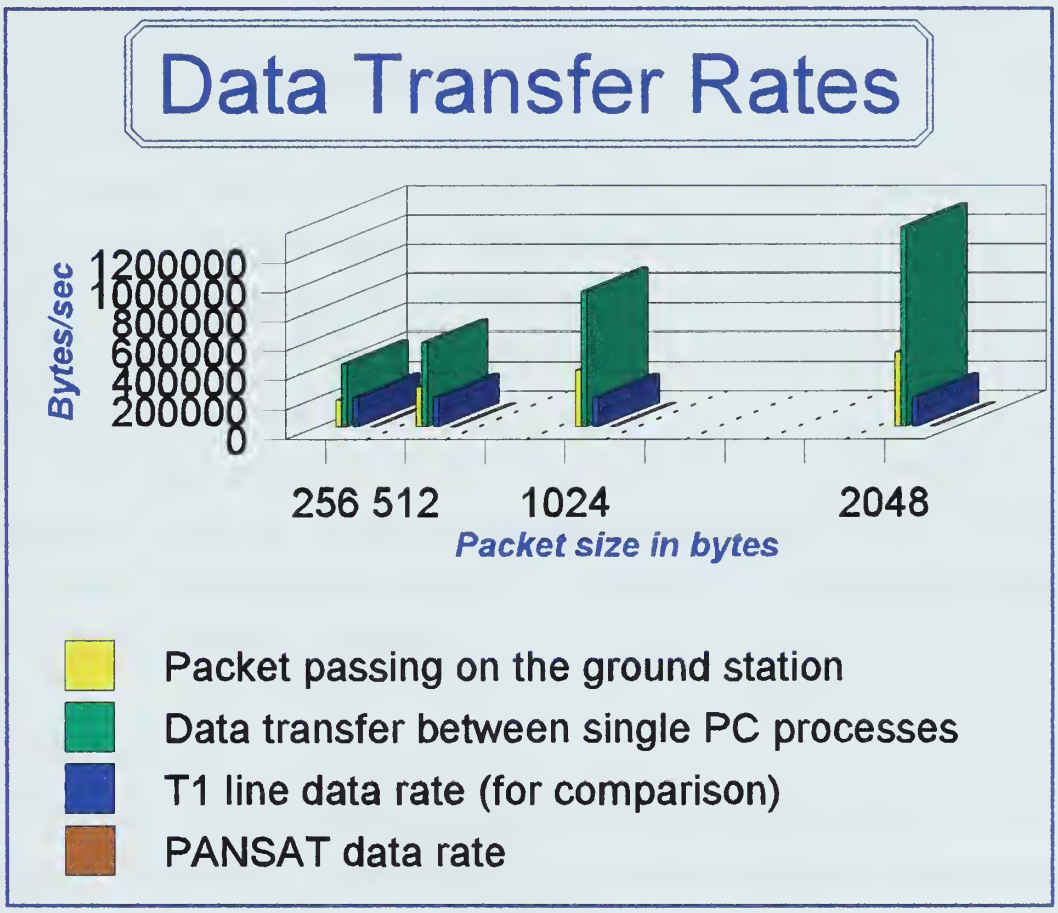


Figure 10 - Real Time Data Processing Testing Results

The results prove that the ground station in the configuration described in this thesis can easily handle the maximum load offered by PANSAT and still have enough leeway to perform all of the ground station processing as well. Using a 256 byte packet size, the response greatly exceeds the goal of two times PANSAT's data rate. In fact, the ground station processing rate is nearly 100 times PANSAT's data rate, almost equaling a full data load supplied by a T1 line. By using larger packet sizes, even a load comparable to that on a T1 line is exceeded. Processing the entire ground station on one PC is significantly faster than a distributed operation, but this is not even a factor since the network performs so well.

Most likely changing the network configuration will not become an issue because of the vast difference between processing capability and data transmission rates. However, if the network configuration of the ground station is significantly modified in the future, these timing tests should be repeated to ensure the new configuration still handles the work load in real time.

The Space Systems Academic Group conducted their own test to ensure that the spacecraft's processor could manage the maximum anticipated data rate, 9842 bps. Although not formally documented, the results indicated that PANSAT's processing capability was sufficient to handle the maximum communications load and still be able to perform all the required systems processing.

Although not projected at this time, if the satellite were to stop operations in spread spectrum mode and use narrow band communications, the data rate would jump up to 78,125 bps. This eventuality would required reevaluation of the load bearing capacity for both the ground station and satellite.

B. TEST PLAN

To ensure that each module is functional and as error free as possible, rigorous testing must be performed on the software before it becomes operational. Following is the script by which programs are to be tested. These scripts attempt to ensure all the basic functionality of the program is operational and that the boundary conditions do not result

in unpredictable behavior. Unfortunately, it is impossible to eliminate all errors from a program, especially one the size of the PANSAT project. Thus the software's performance must be continually monitored by the ground station, which will generate corrections as warranted.

1. Satellite Module

Although a simulator board for a PC exists which emulates the processor of the satellite, this is not sufficient for a full system test of the User Services software. The simulator board does not have a file system or access to storage banks, which are required elements for the program. Therefore, testing can only be accomplished on a fully functional PANSAT platform.

Fortunately, the Space Systems Academic Group is constructing a duplicate PANSAT satellite for testing purposes. Once this satellite is complete and connected to the ground station, either by direct physical connection or by radio transmission, testing can commence.

For testing purposes, instead of using the User Services ground station software, a simple ASCII repeater should be used. The only stipulation is that the data sent from the ASCII repeater must be encoded into an AX.25 packet before it is sent to the satellite. This way, an error with the ground station software will not be misinterpreted as an error within the satellite software, a confusion which could occur in testing two separate packages simultaneously. Several sessions of the ASCII repeater should be brought up simultaneously, each one representing an independent user communicating with PANSAT. For the purposes of this test, the callsign of the first repeater will be "A," the second repeater will "B," and so on.

Since the testing satellite is ground based, the window of opportunity to communicate with PANSAT never actually expires. However, to perform an accurate test, that factor needs to be accounted for. Therefore, the operators conducting the test will have to time themselves. After a suitable time, no further data should be sent to the test platform and any data transmitted from the platform should be ignored.

The script for the spacecraft module testing is very explicitly detailed. Normally, an interface will be wrapped around the communications between a user and PANSAT. However, the testing at this level is conducted with a minimal testing-only interface. The goal is to merely test the functionality of the satellite - the interface will be incorporated and tested with the ground station software. Therefore, the following testing indicates exactly when and what to type:

Test	Command	Result
Connection to PANSAT	C PANSAT A (note that in this case, the originators callsign must be explicitly stated. Once using the ground station software, this will be done automatically)	PANSAT should return series of initial log on messages, broadcast message, and menu of choices
Rejection of too many connections (each one of the connection commands is performed from a different thread on the testing machine)	C PANSAT B C PANSAT C ... C PANSAT O C PANSAT P	Each one of the connections through O should be accepted, connection P should be rejected
Test ground station being able to connect on top of full connection load (this should be done from P's failed thread, since it should be available)	C PANSAT KD6CXV	Connection should be accepted, as described above, with additional print out of last time ground station was connected, which should be never.
Test autodisconnect for users that do not transmit for two minutes	None for several minutes	Each one of the connections should see a disconnection message
A sends an e-mail to B and C (switch to A thread) (note that after the connection and until disconnection, the thread automatically uses the connection callsign for all the remaining commands on the testing platform)	C PANSAT A SM B C S:Test This is a Test message^D	After the connection messages are viewed, the send mail command is entered, after which the menu should be redisplayed (In fact after all the commands performed in this test, the menu should be listed after completion unless otherwise stated)
From A, display all the e-mails on PANSAT	LM E	Should list the one e-mail
Display any e-mail for A	LM U	No e-mails should be listed

Display any e-mail for B (switch to B thread)	C PANSAT B LM U	After the connection messages are displayed, the one e-mail should be listed
Read the e-mail for B	RM # (where # is the e-mail number obtained from the listing above)	"This is a Test message" should be displayed to user
Delete the e-mail for B	DM #	No error messages should be seen
List any e-mail for B	LM U	No e-mails should be listed, as it was deleted in the previous command
List any e-mail for C (switch to C thread)	C PANSAT C LM U	After the connection messages are displayed, the one e-mail should be listed
Delete the e-mail for C	DM #	No error messages should be seen
List any e-mail for C	LM U	No e-mails should be listed
List any e-mail on the system	LM E	No e-mails should be listed, as all recipients have deleted the e-mail
A sends an e-mail to all (switch to thread A)	SM all S:Test2 This is a second Test message^D	After sending, no results should be displayed on the screen
A sends an e-mail to B	SM B S:Private This is a third Test message^D	After sending, no results should be displayed on the screen
List any e-mails for C (switch to C thread)	LM U	No e-mails for C should be listed
List any e-mails for all	LM A	"Test2" e-mail should be listed
List any e-mails for B (switch to B thread)	LM U	"Private" e-mail should be listed
List any e-mails for all	LM A	"Test2" and "Private" e-mails should be listed
List e-mail ranges	LM E # - (where # is the number of the "Private" e-mail)	Only "Private" e-mail should be listed

List e-mail ranges	LM E - # (where # is the number of the "Test2" e-mail)	Only "Test2" e-mail should be listed
See who is currently connected to PANSAT	W	A, B, and C should be listed
Send a one-liner message to A	M A One-Liner Test	"One-Liner Test" should be displayed in A's thread window
Send a one-liner message to all	M all One-Liner Test	"One-Liner Test" should be displayed in A, B & C's thread window
Forward e-mail to C	FM C # (where # is the number of the "Test2" e-mail)	After sending, no results should be displayed on the screen
Check disconnect of A (switch to A thread)	X	A should get a disconnect message
Verify disconnect	LM E	An error message should be displayed
Verify e-mail was forwarded (switch to C thread)	LM U	The forwarded "Private" e-mail should be displayed
Get Help	?	A short description of all the general user BBS commands should be listed out
Attempt to delete an all message	LM A (to get number of Test2 e-mail, which is used for # in next command) DM #	An error message should be displayed not allowing deletion
Verify message not deleted	LM A	Both "Private" and "Test2" e-mails should be listed
Have originator delete the all message (switch to A thread)	C PANSAT A DM # (where # is the number of the "Test2" e-mail)	No error messages should be displayed
Verify deletion	LM E	Only the "Private" e-mail should be listed
Verify originator removing e-mail from system	DM # (where # is the number of the "Test2" e-mail)	No error messages should be displayed
Verify deletion	LM E	No e-mails should be listed

Get current telemetry (switch to B thread)	TC	Current telemetry values should be listed out
Get stored telemetry	TS	A file download should occur, which will contain several days of telemetry (this file needs to be constructed on PANSAT)
Check maximum e-mail length	SM A S:BigTest (repeat a character for over 4096 times) ^D	An error message should be returned, indicating e-mail message was too big
Check e-mail was not saved	LM E	"Test2" should be the only e-mail on the system
Disconnect remaining sessions before continuing	X (from A, B & C)	Each should receive a disconnection message
Connect as ground station (switch to ground station thread)	C PANSAT KD6CXV	Should see connection messages, with last ground station connection time of the beginning of this test
Verify current connections on PANSAT	W	KD6CXV should be only entity listed
List all e-mails on system	LM E	"Test2" should be the only e-mail on the system
Delete an e-mail not destined for the ground station	DM # (where # is the number of the "Test2" e-mail)	A query of eight integers should be sent to the ground station (these are verification numbers - for security reasons the algorithm to answer the query has not been published but separately handed over to the SSAG, which should have implemented a quick calculation routine to determine the answer)
Provide the wrong answer	# (where # is a wrong answer)	An error message should be displayed
Verify e-mail was not deleted	LM E	"Test2" should still be listed
Delete an e-mail not destined for the ground station, correctly this time	DM # (where # is the number of the "Test2" e-mail) # (where # is the correct answer to the query)	After the eight integers are displayed and the correct response is sent to the satellite, no reply should be seen

Verify e-mail was deleted	LM E	No e-mails should be listed
Change broadcast message	P This is a new broadcast message^D # (where # is the correct answer to the eight integer query)	After receiving eight integers for a query, there should be no further messages
Test broadcast message (switch to A thread)	C PANSAT A	The normal connection messages should be displayed, with the addition of the new broadcast message "This is a new broadcast message"
Send an e-mail to ground station	SM B KD6CXV S:ToGS This is even another test message^D	No error messages should be displayed
Send an e-mail just to B	SM B S:ToBee This message is only for B^D	No error messages should be displayed
Check all e-mails on PANSAT (switch to ground station thread)	LM E	Both "ToGS" and "ToBee" e-mails should be listed
Check e-mails to ground station	LM U	Only "ToGS" should be listed
Delete e-mail to ground station	DM E	No error messages should be displayed (Note no verification query should be sent)
Verify e-mail deleted	LM E	Only "ToBee" should be listed
Terminate program	KI # (where # is the correct answer to the query from PANSAT)	After the eight integer query is received and answered, a message to all connected users (A & the ground station) should be sent indicating the program is terminated, then the connection should automatically be disconnected
Verify program exited (switch to A thread)	C PANSAT A	No response should be received - none at all
After reloading User Services software to PANSAT, verify that file structure was maintained	C PANSAT A LM E	After the connection messages are displayed, the e-mail "ToBee" should be listed

Attempt to change the ground stations callsign (switch to ground station thread)	C PANSAT KD6CXV GS newcall	After the connection messages are displayed, the GS operation should result in an error message - the callsign was not sent in triplicate
Change ground station callsign	GS newcall newcall newcall # (where # is the correct answer to the query from PANSAT)	After the eight integer query is sent and answered, a message saying "ground station callsign changed to GS"
Disconnect ground station	X	Disconnection message should be displayed, with reminder ground station callsign is changed and will be effective on the next connection
Connect to PANSAT with old ground sation callsign	C PANSAT KD6CXV	Normal connection messages should be displayed, without last ground station connection time listed
Attempt ground station command	KI	Error message should be sent indication command is reserved for ground station
Disconnect	X	Normal disconnection message
Connect with new ground station callsign	C PANSAT newcall	Connection messages should be displayed including the last time the ground station was connected
Do a ground station only command to get BBS settings	G # (where # is the correct response to PANSAT's query)	After the eight integer query is sent and replied to, a stream of numbers should be displayed
Disconnect ground station	X	Normal disconnection messages should be shown

A sends an file to B and C (switch to A thread) (Although A was not previously disconnected, most likely at this point more than two minutes most likely has elapsed since activity on A occurred - A should have been automatically disconnected - if not, ignore the connection command)	C PANSAT A SF B C File1 (the tester program needs to now send the contents of the file to PANSAT) (Note that unless specifically stated, all test files need to be under 256 kB)	After the file is saved, PANSAT will send a file status report indicating how many bytes were stored
Display all the files on PANSAT	LF E	Should list the one file
Display any files for A	LF U	No file should be listed
Display any files for B (switch to B thread)	C PANSAT B LF U	After the connection messages are displayed, the one file should be listed
Download the file for B	RF # (where # is the file number obtained from the listing above, the ASCII repeater will now have to save the contents of the download in a file)	The file should be downloaded, after which a operating system provided file compare command should be used to ensure the file is the same as the original
Delete the file for B	DF #	No error messages should be seen
List any files for B	LF U	No files should be listed, as it was deleted in the previous command
List any files for C (switch to C thread)	C PANSAT C LF U	After the connection messages are displayed, the one file should be listed
Delete the file for C	DF #	No error messages should be seen
List any files for C	LF U	No files should be listed
List any files on the system	LF E	No files should be listed, as all recipients have deleted the e-mail
A sends a file to all (switch to thread A)	SF all File2	After sending, no results should be displayed on the screen

A sends a file to B	SF B File3	After sending, no results should be displayed on the screen
List any files for C (switch to C thread)	LF U	No files for C should be listed
List any files for all	LF A	File2 should be listed
List any files for B (switch to B thread)	LF U	File3 should be listed
List any files for all	LF A	File2 and File3 should be listed
List file ranges	LF E # - (where # is the number of File3)	Only File3 should be listed
List e-mail ranges	LM E - # (where # is the number of File2)	Only File2 should be listed
Forward file to C	FM C # (where # is the number of File3)	After sending, no results should be displayed on the screen
Verify file was forwarded (switch to C thread)	LF U	The forwarded File3 should be displayed
Check maximum e-mail length	SF A BigFile (BigFile should be over 256 kB in size)	An error message should be returned, indicating file message was too big
Check file was not saved	LF E	File2 and File2 should be the only files on the system
Ensure mail-only delete option does not work on files	DF E	An error message saying option not available for files should be returned

Note that if during the testing a connection to PANSAT is disconnected due to no activity for over two minutes, merely reconnect the thread window to the satellite the next time activity is required by the callsign as indicated in the script.

The remainder of the features of PANSAT can only be tested once the ground station is operational. Therefore, the features not tested in this section will be incorporated into the tests for the ground station.

2. Ground Station Module

In testing the ground station, the scenario presented in the spacecraft module's testing should be replicated, with the following exceptions. First, instead of using the ASCII repeater, the actual ground station software should be used. Second, only the communications terminal should be able to send data to the PANSAT replica. This will happen naturally if the test platform is not physically connected to the ground station, but is rather communicated with via radio transitions. However, if the replica is directly connected to the network, it should be programmed to ignore any data packets except for those sent explicitly to PANSAT from the communications terminal.

While some individual elements of the ground station software package can be tested individually, most of them need to interact with a satellite in order to function correctly. Thus, ground station testing should commence after the spacecraft module has been fully tested. This way, errors in the spacecraft software will not be mistaken for being in the ground station software.

Unlike the spacecraft module, the ground station testing script does not explicitly indicate what keystrokes to type. Rather, the action required is listed. Part of the test is for the user to be able to easily determine the means to accomplish this action. Thus, not only the functionality of the software it tested, but the interface to that functionality is evaluated as well. The script for the ground station testing is as follows:

Module		Test/Action
Control Terminal Monitor	1.	Once the Monitor activates, launch the other four programs shown in the Monitor window.
	2.	Ensure that the status for each program is active and that the action button for each has changed to terminate.
	3.	Using the action button, terminate any one program from monitor. Ensure the status and action button label change accordingly to what they were before the program was originally launched.

Control Terminal Monitor (cont.)	4.	Now choose another program and terminate that program from within itself by clicking the "X" in the top right hand corner. Once again, ensure that the status line and the action button for that program are updated in the Monitor window.
	5.	Enter super user mode using a password that has been separately provided to the SSAG personnel.
	6.	Do nothing for ten minutes. Super user mode should expire.
	7.	Reenter super user mode, then try to exit it manually. The results should be the same as when time expired.
	8.	First change the data in one of the two remaining operating programs, but don't save it - thus when it terminates it should prompt the user. Next, terminate the ground station by pressing that button in the Monitor window. All programs should exit gracefully, including prompting the user to save data.
Control Panel	9.	Visually inspect the displayed settings. They should match the default values indicated in Table 3 (on page 63).
	10.	Modify a couple of the values, ensuring the new parameters are within the bounds listed. Press the send settings to PANSAT button in the Terminal window.
	11.	Change the values once again. However, this time do not send the values to PANSAT. Instead, press the get BBS settings button in the Terminal window. The Control Panel window should now display the values made after the first change.
	12.	Enter several meaningless values, ensure that the errors are caught before a transmission is attempted.
Batch Editor	13.	Create a new batch file.
	14.	Using the command dialog, make a batch file that connects to PANSAT, reads the current telemetry values, then disconnects.
	15.	Save the file, then compile it. Resave the file.
	16.	Select the new batch on batch list by checking the box next to the file name.
	17.	Wait, when the satellite tracking window indicates PANSAT enters the ground station's window of opportunity for communications, ensure communications window on the display terminal shows that batch executed.
	18.	Create a new batch file.
	19.	Attempt to make random illegal commands by typing them in. Compile the program, fixing the errors as they are pointed out. Ensure all errors are detected.
	20.	Now try to insert illegal commands using command dialog. No illegal commands should be able to be entered.

Terminal	21.	Connect to PANSAT in normal user mode (set by Monitor program).
	22.	Generate and send an e-mail Test1 to all.
	23.	Generate and send an e-mail Test2 to A.
	24.	Generate and send e-mail Test3 to the ground station.
	25.	Generate and send e-mail Test4 to the ground station and A.
	26.	Get a listing for all e-mails for the ground station. Test3 and Test4 should be listed.
	27.	Get a listing of all the e-mails sent to the special callsign all. Test1, Test3 and Test4 should be listed.
	28.	Get a listing of all e-mails on PANSAT. All four Test e-mails should be listed.
	29.	Read the Test3 e-mail.
	30.	Delete the Test3 e-mail.
	31.	Get a listing of all the e-mail for the ground station. Only Test4 should be listed.
	32.	Get a listing of all the e-mails on PANSAT. Test1, Test2 and Test4 should be listed.
	33.	Send a one-liner message to all. It should be displayed in the Terminal window.
	34.	Send a one-liner message to the ground station. It should be displayed in the Terminal window.
	35.	Upload a file to PANSAT with the ground station as the recipient. It should be less than 256 kB.
	36.	Get a file listing of all files on the system. Only the single file should be shown. Then download the file and do a file compare with the original.
	37.	Delete the file, do a listing to ensure that it was properly removed.
	38.	Upload another file, but terminate the program in the middle of transmission.
	39.	Reconnect to PANSAT, and when prompted, continue the upload. Then download the file and compare it to the original.
	40.	Download the stored telemetry file.
	41.	Attempt to post a new broadcast message. The try should be unsuccessful.
	42.	In the Monitor window, enter super user mode. Now try to post the new broadcast message again. This time the function should be successful. The verify function handler in the ground station should have intercepted the query by the satellite and answered it automatically.
	43.	View the descriptive help files.
	44.	Repeat this entire section using NPSterm (steps 21 - 43).
Control Archive Manager	45.	Attempt to view the file with the latest telemetry values.
	46.	Attempt to view the current system directory. All the e-mails and files from the terminal test should be listed.
	47.	Conduct a search for an e-mail to the ground station. Two e-mails should be returned.
	48.	Print one of the two e-mails to ensure that the interface to the print manager is functioning.

Control Archive Manage (cont.)	49.	Delete one of the two e-mails. It should be put into the recycle bin on the Server terminal.
	50.	View the file contents of the e-mail that was not deleted.
Display Terminal Monitor	51.	Repeat steps for Control Terminal Monitor (steps 1 - 8), except for the super user setting, which is not available on this terminal.
Telemetry Display	52.	Verify the data displayed is the same as displayed in the archive test performed above. The data should match and the representation should match specification correctly (color coding).
	53.	In the Terminal window, press the download current telemetry button. The Telemetry Display window should update its display automatically when the new data arrives.
Communications Repeater	54.	This test is actually performed when doing the Terminal test.
	55.	When the Terminal test is ongoing, have another tester monitor this window to ensure the entire activity displayed on the Terminal output is identically displayed here.
Display Archive Manager	56.	Perform the identical tests conducted on the Control Archive Manger, then do the following additional tests:
	57.	Ensure file locking is working by opening a file for edit on the Control Terminal, the trying to delete the same file from the Display terminal. An error message should display and the file should not be deleted.
	58.	The same should happen if modifying an already open file, it should be labeled as read only and cannot be saved.
Satellite Tracking	59.	First, the window merely needs to be visually checked to ensure active tracking is occurring. A highlighted window of opportunity needs to include the ground stations and the satellite position indicator needs to move in real time. The accuracy of the tracking can be checked with an orbital program maintained by the SSAG.
	60.	Secondly, an updated initial position needs to be put into the Control Panel module, then transmitted. The Satellite tracking window needs to update its position to this new data as well.
Internet Web Site	61.	The following tests should be performed on a computer with access to the Internet, but not directly connected to the SSAG subnet, as the ground station is.
	62.	Connect to the ground station's Internet home page, IP address http://131.120.25.124/.
	63.	Search through the archives and display the data in any archive file.

Internet Web Site (cont.)	64.	Get the current system directory. Compare the directory to the files listed today's archive files. All the binary files on the satellite should be included. However, only one of the e-mails listed should be included.
	65.	Generate an e-mail and send it. Now from the Terminal window, generate another e-mail and send it. From the Communications Repeater window, verify that both e-mails get sent to PANSAT.
	66.	While the e-mails are being sent, have another person generate an e-mail and send it. This third e-mail should not interfere with the original e-mails. Once again, at the Terminal window, generate and send another e-mail. Ensure that both e-mails get sent to PANSAT.
	67.	From the Terminal Window, send up a file to PANSAT. Then perform a file listing. After this is completed, ensure that the system directory has been updated to include the new file. The new file should not be indicated that it is locally saved, however. Request the file be download. After a small delay, ensure that the file is downloaded from PANSAT (since the ground station's connection with the satellite should be ongoing).
	68.	Send an e-mail to the ground station with the first line formatted "Send: <>", where <> is your Internet e-mail address. From the Terminal window, send an arbitrary e-mail, the send a get all e-mails command. After a reasonable delay, ensure the e-mail originated via the web page was forwarded to your Internet e-mail site.

C. PROGRAM SETUP

This section describes how to install, organize and execute the programs that comprise the User Services package for PANSAT.

1. Satellite Module

The User Services program for the satellite is contained in a single executable file, called US.exe. To upload the software to PANSAT, a BIOS level connection needs to be made with the satellite from the ground station. After the SCOS has been loaded, the command "scload US" is entered. This command sends the program file up to the spacecraft. Once loaded, the program automatically begins running and User Services becomes available for use.

If a new version of User Services becomes available, the ground station must send the "KI" command to the User Services program operating on the satellite. This command saves all the program data and gracefully terminates the program. Once this occurs, a BIOS level connection can be established and the new executable can be uploaded, as described in the previous paragraph.

2. Ground Station Module

On the hard drive of each one of the terminals of the ground station, the directory "PANSAT" needs to be created under the root directory. Under this directory, a "BIN" subdirectory needs to be created. All the executable and help files for each terminal are placed in the "BIN" subdirectory, which is where the ground station software "expects" them to be. If they are not located as such, the program will not work correctly.

Further, on the control terminal, a "BATCH" subdirectory under PANSAT needs to be created. All the batch files, including the system default, need to be located in this subdirectory or they will not be found. All further subdirectories under the "PANSAT" directory will be automatically created as required.

On the communications terminal and server, the directory "ARCHIVE" will be created under the root directory if it does not already exist. No setup is required for this directory, the ground station programs will manage this directory and any subdirectories it creates under it automatically. See Figure 11 for a representation of the directory structure required at installation.

On the desktop of each of the Window NT terminals, a shortcut needs to be made to that terminal's corresponding monitor program. For instance, on the control terminal, a shortcut to the "PANSAT\BIN\ControlTermMonitor.exe" program should be made. These are the only shortcuts required. To start the ground station program for each terminal, the icon on the desktop is simply double clicked. Initial setup is automatically performed by the program, with options to change default settings available in each program. From inside the monitor program, each of the other modules for that particular terminal can be started, dismissed, or controlled.

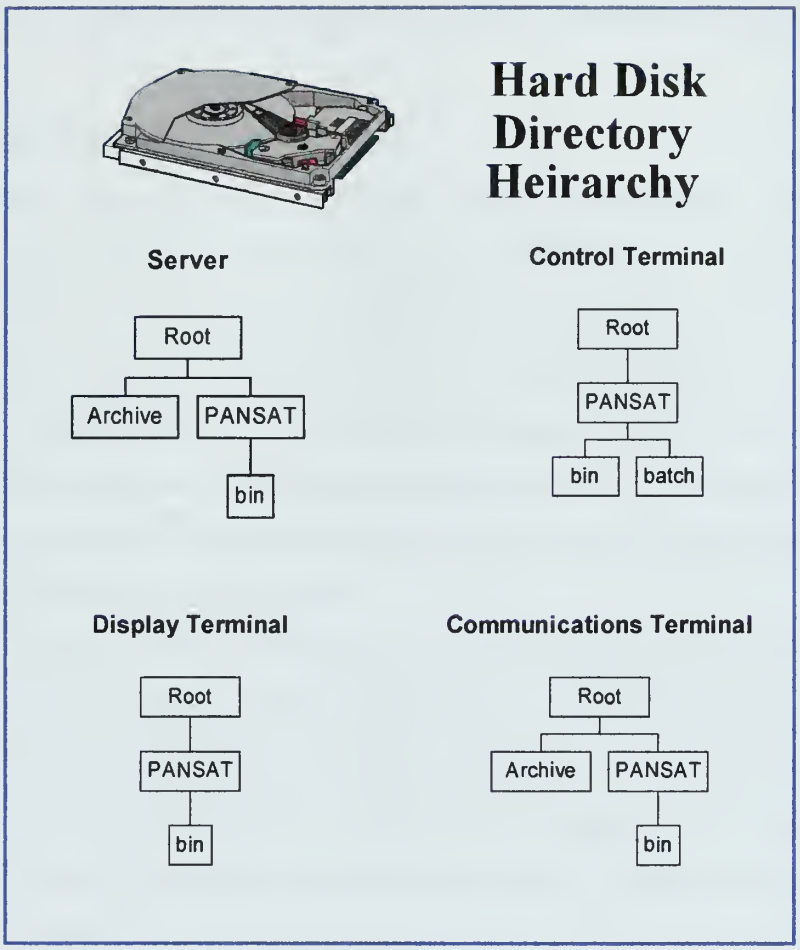


Figure 11 - Ground Station Initialization Directory Structure

The monitor program is the essence of a terminal's ground station activity. While some of the modules do not need to be active for the terminal to perform its required processing, once the monitor program is terminated, the terminal ceases to function as a component of the ground station.

Once these steps have been completed, the ground station should be setup and fully operational.

VIII. CONCLUSIONS AND RECOMMENDATIONS

A. FURTHER WORK REQUIRED

There still exists work to be done on the User Services package. Unfortunately, due to time constraints, only a subset of the programs defined in this thesis could be implemented. While a working product which performs the basic functionality was completed, follow-on work needs to be carried out to finish the project. Future work should start by completing the program to the specifications detailed in this thesis.

After the User Services program is completed, further experiments not elaborated in this thesis could easily be integrated as extra modules into the existing software. Some of the areas that should be considered are:

- Security - if the knowledge gained from PANSAT were to be applied to a larger scale project for the military, the system would have to be known to have the capability to be secure. A system needs to be developed which guarantees the privacy of e-mail to their recipients. Also the means of identifying the ground station either needs to be proven secure or improved.
- Network development - the distributed system of four computers comprising the ground station is not proven to be implemented in the most efficient manner. A study could examine the work load on the different machines, the network traffic between them, and the means of passing the network traffic to determine if the implementation can be improved.
- Interactive web site design - the web site described in this thesis was not developed with a full knowledge of all the capabilities of web technology. An improved web site could be designed with more powerful applications, quicker response, better organization, and easier interface for the end user.
- Human-computer interface improvements - The interface on the Control and Display terminals was not tested to determine if it is intuitive, efficient,

and "user friendly." A study could determine a better methodology for interaction or a better paradigm for interface.

- Communications improvements - as mentioned in the section on NPSterm, only the rudimentary elements are implemented in the protocol. The application layer protocol could be improved with advanced techniques and experimental ideas to provide even more optimized performance.
- Operating system development - SCOS, the current operating system on PANSAT is proprietary and may not support all the future activities requested of the satellite. An in-house developed operating system could be tailor to be PANSAT optimal, and adjusted as many times as necessary to fulfill future needs.
- Experimental features - Just as the positional awareness feature provides new functionality not hereto before seen on a micro satellite, many new features may be put on PANSAT which this author cannot even conceive. Almost *any* future ideas could be attempted on the satellite, as coordinated with the SSAG.

B. LESSONS LEARNED

The biggest lesson learned from developing the User Services was the complexity and diversity of knowledge required for the project, which was initially underestimated. For example, in preparing for the ground station programming, the author realized the need to learn the Windows NT development environment. While correctly identifying the need to learn event-driven programming paradigm and a Win32 kernel accessing framework (Microsoft Foundation Classes, in this case), the actual time required to learn these concepts was not realized. An estimation of six weeks turned into nearly six months used to cover enough concepts to implement the intricate requirements of the ground station. Future work on the project should identify the concepts required to implement the feature, then budget enough time to research the concepts. Along the same lines,

future work should limit itself in scope so as to not try to cover too many topics at once, which becomes overwhelming.

A characteristic which would increase the software development resources of the SSAG would be stronger ties to the Computer Science (CS) Department. The author of this thesis almost accidently got involved with the SSAG. In fact, most students in the CS curriculum are unaware of PANSAT project or even SSAG's existence. Those that did become aware were generally already involved with their own research or predisposed to a field of study precluding SSAG due to late exposure in their matrix. Hopefully future projects for PANSAT can get the CS faculty involved, increasing that departments cognizance of the research possibilities with the SSAG. Exacerbating this situation, the SSAG's software section consists of only one individual, which is only a fraction of the manpower necessary to perform the work requirement in software development for the PANSAT project.

Finally, as evidenced by this thesis being the first formal specification for the bulk of PANSAT's software, software engineering did not play an early enough role in the development of the satellite. Hardware design took place long before software consideration began. This, unfortunately, cuts off a valuable resource in the development of the overall system. If possible, in the future, software engineers should assist the hardware developers at an early stage to allow for design considerations which would aid the software in fulfilling the overall system goals.

C. CONCLUSION

This thesis provides the necessary documentation to formally specify the communication service software requirements and functionality. Additionally, several satellite experiments are defined, as well as the means to conduct further experiments offered. The definition is much more complex and encompassing than originally anticipated, not only qualifying the internal details of several systems, but the entire external interactions of those systems as well .

The foundation of the PANSAT project is an educational tool. This basis means that the scope of the User Services software will most likely change in the time to come. For all future work on the User Services, experimental features, or ground station implementation, this thesis forms the framework. Any modifications, however slight, should refer to the outline herein to identify the scope of the code to be modified, thus preventing unwanted side effects.

APPENDIX A. SELECTED SOURCE CODE EXTRACTS

Due to the enormous size of the source code, it is not included in this thesis publication. Rather only the portions which implement the experimental features are included. The entire program source can be obtained from myself (gkhunter@msn.com) or Jim Horning (JHorning@nps.navy.mil).

A. POSITION DETERMINATION CODE

This program was referred to in Chapter 6, Section A.

File: orb_cal.h

```
/*************************************************************************
/*************************************************************************
/ File: orb_cal.h
/ Operating Environment: SCOS
/ Compiler: Microsoft C ver 5.0/5.1
/ Last Modified: 01 MAR 98
/
/ Description:
/    - This module returns the latitude and longitude of the satellite
/      based on a Keplerian Element initial position
/    - The initial position is provided by the ground station and is
/      periodically updated
/      - The calculated position is based on the time the calculation
/        is performed
/    - The function returns a structure with two real numbers
/      - A positive latitude means north
/      - A positive longitude means east
/    - This program is based on the public domain program by Karl
/      Meinzer and James Miller, modified to perform spacecraft
/      specific calculations
/*************************************************************************
/*************************************************************************/
```

File: orb_cal.h (cont.)

```
/* Structure for satellite element set - input for the function */
struct sat_elements {
   double epoch_yr;
   double epoch_day;
   double inc;
   double raan;
   double eccen;
   double arg_peri;
   double m_anomaly;
   double m_motion;
};


/* Structure for saving calculated satellite
   position - results of the function */
struct sat_position {

   double lat;
   double lon;

};


/* based on sat_elements input, returns the current position
   of the satellite in lat/long in sat_postion */
void calculate_pos(struct sat_elements *, struct sat_position *);
```

File: orb_cal.c

```
/*****************************************************************************
/*****************************************************************************
/ File: orb_cal.c
/ Operating Environment: SCOS
/ Compiler: Microsoft C ver 5.0/5.1
/ Last Modified: 01 MAR 98
/
/ Description:
/    - This module returns the latitude and longitude of the satellite
/      based on a Keplerian Element initial position
/    - The initial position is provided by the ground station and is
/      periodically updated
/      - The calculated position is based on the time the calculation
/        is performed
```

File: orb_cal.c (cont.)

```
/     - The function returns a structure with two real numbers
/       - A positive latitude means north
/       - A positive longitude means east
/     - This program is based on the public domain program by Karl
/       Meinzer and James Miller, modified to perform spacecraft
/       specific calculations
/
/ Input: Keplerian Elements in the sat_elements structure
/
/ Output: Lat/Long in the sat_position structure
/
/ Process: - Get the current time
/          - Project Earth position to the current time based on the
/            known position on Jan 1, 1978, at 0000.
/          - Project the satellite's orbit position from the given
/            position to the current time.
/          - Map the satellite's coordinates to a solar system fixed
/            coordinate system
/          - Map the solar system fixed coordinate system to the Earth's
/            coordinates
/          - Return the results.
/
/ Assumptions: The floating point module needs to be linked with this
/              function in order to produce a working program
/
/ Warnings: None
/*************************************************************************
/*************************************************************************/


/*-----------------------------------------------------------*/
/*------------------ Included Headers ---------------------*/
/*-----------------------------------------------------------*/
#include <stdlib.h>    /* for atoi */
#include <math.h>
#include <string.h>
#include <time.h>
#include <sys\timeb.h> /* for timeb struct */

#include "orb_cal.h"


/*-----------------------------------------------------------*/
/*------------- Orbital Mechanic Constants ----------------*/
/*-----------------------------------------------------------*/
#define PI      3.14159265359
```

File: orb_cal.c (cont.)

```
/* Siderial starting point-1978 Jan 01 @ 0000 utc */
#define GHAA    100.29
#define EROTRAT 360.985647 /* Earth's rotation rate in deg/day */
#define ERADIUS 6378.0     /* Earth's radius in km */
#define MINDAY  1440       /* Minutes per day */
#define DAYYR   365.25     /* Days per year */
#define DAYMO   30.6       /* Days per month */
#define HRSDAY  24         /* Hours per day */
#define DAY1978 28125      /* Days since 1978 Jan 01 @ 0000 utc */
/* Numerical value of G * 4 * Earth Mass * PI^2 */
#define GM4PI2  331.25


/*-------------------------------------------------------*/
/*------------ Local Function Declarations--------------*/
/*-------------------------------------------------------*/
double current_time (void);
void rad_to_polar (double x, double y, double *p, double *r);
void polar_to_rad (double p, double r, double *x, double *y);
double rad_to_deg (double x);
double deg_to_rad (double x);
double sgn (double x);
double dabs (double x);
void strmid (char str_out[10], char str_in[10], int x, int y);


/*-------------------------------------------------------*/
/*---------------- Function Definitions -----------------*/
/*-------------------------------------------------------*/

/* Public function call to calculate position */
void calculate_pos (struct sat_elements *se_ptr,
                    struct sat_position *sp_ptr) {

   double period, x, y, d, m, n, p, a, u, g, h, q, v, k, r, s, t;

   x = 0;
   y = 0;

   period = MINDAY / se_ptr->m_motion;

   g = floor(DAYYR * (se_ptr->epoch_yr - 1)) -
       DAY1978 + se_ptr->epoch_day -
       se_ptr->m_anomaly / se_ptr->m_motion / 360;

   /*Calculate orbital period*/
   n = se_ptr->m_motion * 2 * PI;
```

File: orb_cal.c (cont.)

```
/* Calculate semi-major axis */
a = GM4PI2 * pow (period, 0.6666666666667) / ERADIUS;

/* Calculate rate of change of Argument of Perigee */
v = 4.97 * pow (a, -3.5) *
    (5 * pow (cos (deg_to_rad (se_ptr->inc)), 2) - 1) /
    pow(1 - pow (se_ptr->eccen, 2), 2);

/* Calculate rate of change of R.A.A.N. */
q = -9.95 * pow (a, -3.5) * cos (deg_to_rad (se_ptr->inc)) /
    pow(1 - pow (se_ptr->eccen, 2), 2);

t = current_time ();
d = t - g;
k = deg_to_rad (d * q + se_ptr->raan - GHAA - t * 360.985647);

m = d * n;
r = se_ptr->eccen;
p = m;

/* Begin calculating True Anomaly and satellite geocenter distance */
do {

   polar_to_rad (p, r, &x, &y);
   h = (m - p + y) / (1 - x);
   p = p + h;

} while (dabs (h) > 0.001);        /*Check for convergence*/

r = 1;
polar_to_rad (p, r, &x, &y);
y = y * sqrt (1 - se_ptr->eccen * se_ptr->eccen);
x = x - se_ptr->eccen;

rad_to_polar (x, y, &p, &r);
r = a * r;
p = p + deg_to_rad (se_ptr->arg_peri + d * v);

polar_to_rad (p, r, &x, &y);
h = x;
r = y;
p = deg_to_rad (se_ptr->inc);

polar_to_rad (p, r, &x, &y);
s = y;
y = x;
x = h;
```

File: orb_cal.c (cont.)

```
   rad_to_polar (x, y, &p, &r);
   k = p + k;
   u = r;

   k = k / 2 / PI;
   k = (k - floor(k)) * 2 * PI;
   x = u;
   y = s;

   rad_to_polar (x, y, &p, &r);
   sp_ptr->lat = rad_to_deg (p);
   sp_ptr->lon = rad_to_deg (k);

   return;

} // end calculate_pos


double current_time (void) {

   struct timeb time_buffer;
   char pos_date[8], pos_time[8], temp[15];
   double hr, min, sec;
   int day, mon, year;

   _strdate (pos_date);
   _strtime (pos_time);

   /* Extract and save month from date string */
   temp[0] = '\0';
   strmid (temp, pos_date, 1, 2);
   mon = atoi (temp);

   /* Extract and save day from date string */
   strmid (temp, pos_date, 3, 2);
   day = atoi (temp);

   /* Extract and save year from date string */
   strmid (temp, pos_date, 6, 2);
   year = atoi (temp);

   /* Extract and save hour from time string */
   strncpy (temp, pos_time, 2);
   hr = (double) (atoi (temp) + time_buffer.timezone / 60);

   /* Extract and save minutes from time string */
   strmid (temp, pos_time, 3, 2);
   min = (double) atoi (temp);
```

File: orb_cal.c (cont.)

```
   /* Extract and save seconds from time string */
   strmid (temp, pos_time, 6, 2);
   sec = (double) atoi (temp);

   /* The following calculates # of days and fraction of a day */
   mon = mon + 1;

   /* since GHAA reference point 1978 Jan 01 @0000 utc. */
   if (mon < 4) {

      year = year - 1;
      mon = mon + 12;

   } // end if

   day = day + (int) floor (year * DAYYR) +
               (int) floor (mon * DAYMO) - 28553;

   return ( hr + min / 60 + sec / 3600) / 24 + (double) day;

} /* end current_time */


/* Rectangular to polar transformation */
void rad_to_polar (double x, double y, double *p, double *r) {

   *r = sqrt (x * x + y * y);

   if (x == 0)
      *p = PI / 2 * sgn (y);
   else
      *p = atan2 (y , x);

   return;

} /* end rad_to_polar */


/* Polar to rectangular transformation */
void polar_to_rad (double p, double r, double *x, double *y) {

   *x = r * cos(p);
   *y = r * sin(p);

   return;

} /* end polar_to_rad */
```

File: orb_cal.c (cont.)

```
/* Return sign of a number */
double sgn (double x) {

   if (x >= 0)
      return (1);
   else
      return (-1);

} /* end sgn */


/* Return absolute value of a double type */
double dabs (double x) {

   if (x >= 0)
      return (x);
   else
      return (x * -1.0);

} /* end dabs */


/* Convert radians to degrees */
double rad_to_deg (double x) {

   return (x * 180 / PI);

} /* end rad_to_deg */


/* Convert degrees to radians */
double deg_to_rad (double x) {

   return (PI * x / 180);

} /* end deg_to_rad */


/* Extract y characters from a string starting at x */
void strmid (char str_out[], char str_in[], int x, int y) {

   int i;

   for (i = x; i < (x + y); str_out[i - x] = str_in[i], i++);
   str_out[i-x] = '\0';

   return;

} /* end strmid */
```

B. COMPRESSION CODE

This program was referred to in Chapter 6, Section C. This is a modification of the LZSS program by Mark Nelson and Jean-Loup Gailly.

File: Lzss.c

```
/*************************************************************************
/*************************************************************************
/ File: Lzss.c
/ Operating Environment: SCOS
/ Compiler: Microsoft C ver 5.0/5.1
/ Last Modified: 01 MAR 98
/
/ Description:
/    - This is the LZSS module, which implements an LZ77 style
/      compression algorithm.  As iplemented here it uses a 12 bit index
/      into the sliding window, and a 4 bit length, which is adjusted to
/      reflect phrase lengths of between 2 and 17 bytes.
/    - This program is a modification of a program by Mark Nelson and
/      Jean-Luc Gaily
/
/ Input: The file to be compressed or decompressed
/
/ Output: The compressed or decompressed file, as directed
/
/ Process: - Open the file, compress/decompress the data
/          - Save compressed/decompressed data in temp file
/          - Copy temp file over original
/          - Maintain compression tables throughout execution lifetime
/
/ Assumptions: None
/
/ Warnings: None
/*************************************************************************
/*************************************************************************/


/*-------------------------------------------------------*/
/*----------------- Included Headers --------------------*/
/*-------------------------------------------------------*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

#include "bitio.h"
```

File: Lzss.c (cont.)

```
/*---------------------------------------------------------*/
/*---------------- Compression Constants ------------------*/
/*---------------------------------------------------------*/

/* Various constants used to define the compression parameters.  The
   INDEX_BIT_COUNT tells how many bits we allocate to indices into the
   text window.  This directly determines the WINDOW_SIZE.  The
   LENGTH_BIT_COUNT tells how many bits we allocate for the length of
   an encode phrase. This determines the size of the look ahead buffer.
   The TREE_ROOT is a special node in the tree that always points to
   the root node of the binary phrase tree.  END_OF_STREAM is a special
   index used to flag the fact that the file has been completely
   encoded, and there is no more data.  UNUSED is the null index for
   the tree. MOD_WINDOW() is a macro used to perform arithmetic on tree
   indices.                                                          */

#define INDEX_BIT_COUNT      12
#define LENGTH_BIT_COUNT     4
#define WINDOW_SIZE          (1 << INDEX_BIT_COUNT)
#define RAW_LOOK_AHEAD_SIZE  (1 << LENGTH_BIT_COUNT)
#define BREAK_EVEN        ((1 + INDEX_BIT_COUNT + LENGTH_BIT_COUNT) / 9)
#define LOOK_AHEAD_SIZE      (RAW_LOOK_AHEAD_SIZE + BREAK_EVEN)
#define TREE_ROOT            WINDOW_SIZE
#define END_OF_STREAM        0
#define UNUSED               0
#define MOD_WINDOW(a)        ((a) & (WINDOW_SIZE - 1))

char *CompressionName = "LZSS Encoder";
char *Usage           = "in-file out-file\n\n";


/* These are the two global data structures used in this program.
   The window[] array is exactly that, the window of previously seen
   text, as well as the current look ahead text.  The tree[] structure
   contains the binary tree of all of the strings in the window sorted
   in order.                                                         */

unsigned char window[WINDOW_SIZE];

struct {
    int parent;
    int smaller_child;
    int larger_child;
} tree[ WINDOW_SIZE + 1];
```

File: Lzss.c (cont.)

```
/*--------------------------------------------------------*/
/*------------- Local Function Declarations--------------*/
/*--------------------------------------------------------*/
void InitTree (int);
void ContractNode (int, int);
void ReplaceNode (int, int);
int FindNextNode (int);
void DeleteString (int);
int AddString (int, int *);
void CompressFile (FILE *, BIT_FILE *);
void ExpandFile (BIT_FILE *, FILE *);


/*--------------------------------------------------------*/
/*---------------- Function Definitions ------------------*/
/*--------------------------------------------------------*/

/* Since the tree is static data, it comes up with every node
   initialized to 0, which is good, since 0 is the UNUSED code.
   However, to make the tree really usable, a single phrase has to be
   added to the tree so it has a root node.  That is done right here. */
void InitTree (int r) {

    tree[TREE_ROOT].larger_child = r;
    tree[r].parent = TREE_ROOT;
    tree[r].larger_child = UNUSED;
    tree[r].smaller_child = UNUSED;

    return;

} /* end InitTree */


/* This routine is used when a node is being deleted.  The link to
   its descendant is broken by pulling the descendant in to overlay
   the existing link.                                              */
void ContractNode (int old_node, int new_node ) {

    tree[ new_node ].parent = tree[ old_node ].parent;

    if (tree[tree[old_node].parent].larger_child == old_node)
        tree[tree[old_node].parent].larger_child = new_node;
    else
        tree[tree[old_node].parent].smaller_child = new_node;

    tree[old_node].parent = UNUSED;
    return;

} /* end ContractNode */
```

File: Lzss.c (cont.)

```
/* This routine is also used when a node is being deleted.  However,
   in this case, it is being replaced by a node that was not previously
   in the tree.                                                        */
void ReplaceNode (int old_node, int new_node ) {

    int parent = tree[ old_node ].parent;

    if (tree[parent].smaller_child == old_node)
        tree[parent].smaller_child = new_node;
    else
        tree[parent].larger_child = new_node;

    tree[new_node] = tree[old_node];
    tree[tree[new_node].smaller_child].parent = new_node;
    tree[tree[new_node].larger_child].parent = new_node;
    tree[old_node].parent = UNUSED;

    return;

} /* end ReplaceNode */


/* This routine is used to find the next smallest node after the node
   argument.  It assumes that the node has a smaller child.  We find
   the next smallest child by going to the smaller_child node, then
   going to the end of the larger_child descendant chain.              */
int FindNextNode (int node) {

    int next = tree[node].smaller_child;

    while (tree[next].larger_child != UNUSED)
        next = tree[next].larger_child;

    return(next);

} /* end FindNextNode */
```

File: Lzss.c (cont.)

```
/* This routine performs the classic binary tree deletion algorithm.
   If the node to be deleted has a null link in either direction, we
   just pull the non-null link up one to replace the existing link.
   If both links exist, we instead delete the next link in order, which
   is guaranteed to have a null link, then replace the node to be
   deleted with the next link.                                          */
void DeleteString (int p) {

    int  replacement;

    if (tree[p].parent == UNUSED)
        return;

    if (tree[p].larger_child == UNUSED)
        ContractNode (p, tree[p].smaller_child);
    else if (tree[p].smaller_child == UNUSED)
        ContractNode (p, tree[p].larger_child);
    else {
        replacement = FindNextNode (p);
        DeleteString (replacement);
        ReplaceNode (p, replacement);
    } /* end if */

    return;

} /* DeleteString */


/* This where most of the work done by the encoder takes place.  This
   routine is responsible for adding the new node to the binary tree.
   It also has to find the best match among all the existing nodes in
   the tree, and return that to the calling routine.  To make matters
   even more complicated, if the new_node has a duplicate in the tree,
   the old_node is deleted, for reasons of efficiency.                  */
int AddString (int new_node, int * match_position) {

    int i;
    int test_node;
    int delta;
    int match_length;
    int * child;

    if (new_node == END_OF_STREAM)
        return (0);
    test_node = tree[TREE_ROOT].larger_child;
    match_length = 0;
```

File: Lzss.c (cont.)

```
    for ( ; ; ) {

        for (i = 0; i < LOOK_AHEAD_SIZE; i++)  {

            delta = window[MOD_WINDOW (new_node + i)] -
                    window[MOD_WINDOW (test_node + i)];

            if (delta != 0)
                break;

        } /* end for */

        if (i >= match_length) {

            match_length = i;
            *match_position = test_node;

            if (match_length >= LOOK_AHEAD_SIZE) {
                ReplaceNode (test_node, new_node);
                return (match_length);
            } /* end if */

        } /* end if */

        if (delta >= 0)
            child = &tree[test_node].larger_child;
        else
            child = &tree[test_node].smaller_child;

        if (*child == UNUSED) {
            *child = new_node;
            tree[new_node].parent = test_node;
            tree[new_node].larger_child = UNUSED;
            tree[new_node].smaller_child = UNUSED;
            return (match_length);
        } /* end if */

        test_node = *child;

    } /* end for (endless) */

} /* end DeleteString */
```

File: Lzss.c (cont.)

```
/* This is the compression routine.  It has to first load up the look
   ahead buffer, then go into the main compression loop.  The main loop
   decides whether to output a single character or an index/length
   token that defines a phrase.  Once the character or phrase has been
   sent out, another loop has to run.  The second loop reads in new
   characters, deletes the strings that are overwritten by the new
   character, then adds the strings that are created by the new
   character.
*/

void CompressFile(FILE * input, BIT_FILE * output) {

    int i;
    int c;
    int look_ahead_bytes;
    int current_position;
    int replace_count;
    int match_length;
    int match_position;

    current_position = 1;
    for (i = 0; i < LOOK_AHEAD_SIZE; i++) {
        if ((c = getc(input)) == EOF)
            break;
        window[current_position + i] = (unsigned char) c;
    } /* end for */

    look_ahead_bytes = i;
    InitTree (current_position);
    match_length = 0;
    match_position = 0;
    while (look_ahead_bytes > 0) {
        if (match_length > look_ahead_bytes)
            match_length = look_ahead_bytes;

        if (match_length <= BREAK_EVEN) {

            replace_count = 1;
            OutputBit (output, 1);
            OutputBits (output,
                        (unsigned long) window[current_position], 8);

        } else {
```

File: Lzss.c (cont.)

```
            OutputBit (output, 0);
            OutputBits (output,
                        (unsigned long) match_position,
                        INDEX_BIT_COUNT);
            OutputBits (output,
                        (unsigned long)(match_length -(BREAK_EVEN + 1)),
                        LENGTH_BIT_COUNT);
            replace_count = match_length;

        } /* end if */

        for (i = 0; i < replace_count; i++) {

            DeleteString (MOD_WINDOW (current_position +
                                      LOOK_AHEAD_SIZE));
            if ((c = getc(input)) == EOF)
                look_ahead_bytes--;
            else
                window[MOD_WINDOW (current_position + LOOK_AHEAD_SIZE)]
                        = (unsigned char) c;

            current_position = MOD_WINDOW (current_position + 1);
            if (look_ahead_bytes)
                match_length = AddString (current_position,
                                          &match_position);

        } /* end for */

    } /* end while */

    OutputBit (output, 0);
    OutputBits (output, (unsigned long) END_OF_STREAM, INDEX_BIT_COUNT);

} /* CompressFile */


/* This is the expansion routine for the LZSS algorithm.  All it has
   to do is read in flag bits, decide whether to read in a character or
   a index/length pair, and take the appropriate action.
*/
void ExpandFile(BIT_FILE * input, FILE * output) {

    int i;
    int current_position;
    int c;
    int match_length;
    int match_position;

    current_position = 1;
```

File: Lzss.c (cont.)

```
    for ( ; ; ) {

        if (InputBit(input)) {

            c = (int) InputBits(input, 8);
            putc (c, output);
            window [current_position] = (unsigned char) c;
            current_position = MOD_WINDOW (current_position + 1);

        } else {

            match_position = (int) InputBits (input, INDEX_BIT_COUNT);
            if (match_position == END_OF_STREAM)
                break;
            match_length = (int) InputBits (input, LENGTH_BIT_COUNT);
            match_length += BREAK_EVEN;

            for (i = 0; i <= match_length; i++) {

                c = window [MOD_WINDOW (match_position + i)];
                putc(c, output);
                window[current_position] = (unsigned char) c;
                current_position = MOD_WINDOW (current_position + 1);

            } /* end for */

        } /* end if */

    } /* end for (endless) */

} /* end ExpandFile */
```

File: Bitio.h

```
/*****************************************************************************
/*****************************************************************************
/ File: Bitio.c
/ Operating Environment: SCOS
/ Compiler: Microsoft C ver 5.0/5.1
/ Last Modified: 01 MAR 98
/
/ Description:
/    - This utility file contains all of the routines needed to
/      implement bit oriented routines.
/    - The utility functions perform file IO bit per bit rather than
/      by the standard byte
```

File: Bitio.h (cont.)

```
/     - This program is a modification of a program by Mark Nelson and
/       Jean-Luc Gaily
/************************************************************************
/************************************************************************/


#ifndef _BITIO_H
#define _BITIO_H


typedef struct bit_file {
    FILE *file;
    unsigned char mask;
    int rack;
} BIT_FILE;


BIT_FILE     * OpenInputBitFile (char);
BIT_FILE     * OpenOutputBitFile (char *);
void           OutputBit (BIT_FILE *, int);
void           OutputBits (BIT_FILE *, unsigned long, int);
int            InputBit (BIT_FILE *);
unsigned long  InputBits (BIT_FILE *, int);
void           CloseInputBitFile (BIT_FILE *);
void           CloseOutputBitFile (BIT_FILE *);
void           FilePrintBinary (FILE *, unsigned int, int);


#endif  /* _BITIO_H */
```

File: Bitio.c

```
/************************************************************************
/************************************************************************
/ File: Bitio.c
/ Operating Environment: SCOS
/ Compiler: Microsoft C ver 5.0/5.1
/ Last Modified: 01 MAR 98
/
/ Description:
/     - This utility file contains all of the routines needed to
/       implement bit oriented routines.
/     - The utility functions perform file IO bit per bit rather than
/       by the standard byte
/     - This program is a modification of a program by Mark Nelson and
/       Jean-Luc Gaily
```

File: Bitio.c (cont.)

```
/
/ Assumptions: None
/
/ Warnings: None
/*************************************************************************
/*************************************************************************/


/*-------------------------------------------------------*/
/*------------------ Included Headers -------------------*/
/*-------------------------------------------------------*/
#include <stdio.h>
#include <stdlib.h>

#include "bitio.h"


/*-------------------------------------------------------*/
/*---------------- Function Definitions -----------------*/
/*-------------------------------------------------------*/
BIT_FILE * OpenOutputBitFile (char * name) {

    BIT_FILE * bit_file;

    bit_file = (BIT_FILE *) calloc (1, sizeof (BIT_FILE));
    if (bit_file == NULL)
        return (bit_file);

    bit_file->file = fopen (name, "wb");
    bit_file->rack = 0;
    bit_file->mask = 0x80;

    return( bit_file );

} /* OpenOutputBitFile */


BIT_FILE * OpenInputBitFile (char name) {

    BIT_FILE * bit_file;

    bit_file = (BIT_FILE *) calloc (1, sizeof (BIT_FILE));
    if (bit_file == NULL)
        return (bit_file);

    bit_file->file = fopen (name, "rb");
    bit_file->rack = 0;
    bit_file->mask = 0x80;
```

File: Bitio.c (cont.)

```
    return( bit_file );

} /* OpenInputBitFile */


void CloseOutputBitFile (BIT_FILE * bit_file) {

    if (bit_file->mask != 0x80)
        if (putc (bit_file->rack, bit_file->file) != bit_file->rack)
            halt (); /* Fatal error in CloseBitFile! */

    fclose (bit_file->file);
    free ((char *) bit_file);

    return;

} /* CloseOutputBitFile */


void CloseInputBitFile (BIT_FILE * bit_file) {

    fclose (bit_file->file);
    free ((char *) bit_file);

    return;

} /* CloseInputBitFile */


void OutputBit (BIT_FILE * bit_file, int bit) {

    if (bit)
        bit_file->rack |= bit_file->mask;
    bit_file->mask >>= 1;

    if (bit_file->mask == 0) {

            if (putc (bit_file->rack, bit_file->file) != bit_file->rack)
                halt (); /* Fatal error in OutputBit! */

            bit_file->rack = 0;
            bit_file->mask = 0x80;

    } /* end if */

} /* end OutputBit */
```

File: Bitio.c (cont.)

```
void OutputBits(BIT_FILE * bit_file, unsigned long code, int count) {

    unsigned long mask = 1L << (count - 1);

    while (mask != 0) {

        if (mask & code)
            bit_file->rack |= bit_file->mask;
        bit_file->mask >>= 1;

        if (bit_file->mask == 0) {

              if (putc (bit_file->rack, bit_file->file) !=
                  bit_file->rack)

                  halt (); /* "Fatal error in OutputBit! */

              bit_file->rack = 0;
           bit_file->mask = 0x80;

        } /* end if */

        mask >>= 1;

    } /* end while */

} /* end OutputBits */


int InputBit (BIT_FILE * bit_file) {

    int value;

    if (bit_file->mask == 0x80) {

        bit_file->rack = getc (bit_file->file);

        if (bit_file->rack == EOF)
            halt (); /* Fatal error in InputBit! */

    } /* end if */

    value = bit_file->rack & bit_file->mask;
    bit_file->mask >>= 1;
    if (bit_file->mask == 0)
        bit_file->mask = 0x80;

    return(value ? 1 : 0);
} /* end InputBit */
```

File: Bitio.c (cont.)

```
unsigned long InputBits (BIT_FILE * bit_file, int bit_count) {

    unsigned long mask;
    unsigned long return_value;

    mask = 1L << (bit_count - 1);
    return_value = 0;

    while (mask != 0) {
        if (bit_file->mask == 0x80) {

                bit_file->rack = getc (bit_file->file);

                if (bit_file->rack == EOF)
                halt (); /* "Fatal error in InputBit! */

        } /* end if */

        if (bit_file->rack & bit_file->mask)
            return_value |= mask;
        mask >>= 1;
        bit_file->mask >>= 1;

        if (bit_file->mask == 0)
            bit_file->mask = 0x80;

    } /* end while */

    return (return_value);

} /* end InputBits */


void FilePrintBinary (FILE * file, unsigned int code, int bits) {

    unsigned int mask = 1 << (bits - 1);

    while (mask != 0) {

        if (code & mask)
            fputc('1', file);
        else
            fputc('0', file);

        mask >>= 1;

    } /* end while */

} /* end FilePrintBinary */
```

APPENDIX B. SOURCE CODE ORGANIZATION

As stated in Appendix A, except for two excerpts, the source code for the User Services in not included in this thesis. It is simply too voluminous, adding several hundred more pages to this document. To obtain copies of the source code, just e-mail a request to LT Ken Hunter at gkhunter@msn.com or Jim Horning at JHorning@nps.navy.mil. Either one will be happy to e-mail anybody a copy of the program listings.

The purpose of this section, however, is to give a brief description of the source code organization and philosophy. This will help in future efforts to modify or add to the existing User Services software.

All of the source code for the User Services programs, both ground station and satellite, are extensively commented. Not only does this aid in readability, it also makes learning the exact behavior of the program easier. Each function or class method is labeled with the required input, the behavior, and the result of the procedure. Additionally, inside each procedure any non-intuitive block of code is labeled with the desired effect and behavior of that block. Finally, each file contains a header summarizing the purpose and interface with the rest of the program for that particular module.

The ground station software is organized into a separate program for each of the modules described in the description. For example, the module to display the telemetry is a separate .EXE file from the archive manager module. On each terminal however, each of the .EXE files is treated as a thread by the monitor program. The monitor program for each terminal launches each of the other programs, as required. It retains pointers to these other programs, however, in order for the monitor to conduct thread-like interaction with the programs it launches. It is possible for a user to explicitly launch one of the non-monitor programs on a terminal without going through the monitor. This is unadvisable since the monitor would therefore be unable to interact with the newly launched program, reducing the functionality of the software. Towards this end, only the monitor program has an icon on the desktop.

The composition of each ground station program is based on the Microsoft Visual C++ View/Document architecture. This architecture is based around the use of the Microsoft Foundation Classes (MFC). Each program is centered around the interaction of four primary classes. The first class represents the program itself and processes anything not handled by the other four classes. The second class represents the visible window. This class processes any interaction with the user outside the scope of the data being handled by the program. The third class is the document class. The document class maintains the data that is being manipulated. For instance, in the telemetry viewing program, the actual telemetry values being displayed are stored in this class. Finally, the view class visually represents the data held in the document class onto the displayed window. Most of a program's interaction is handled by this class since the view also processes input directed towards document manipulation. This is a quick purview of the primary classes used for each one of the User Services programs for the Windows NT operating system. For a detailed description of the View/Document architecture and the intricate interaction between the primary classes, an MFC tutorial should be referenced.

The source code file names are comprised of the program name, followed by the class name. Thus, for the telemetry's view class, the source code file name is "TelemetryView." Using standard C++ format, there are two files with this name, an "H" file and a "CPP" file. The "H" file contains the interface to the class, while the "CPP" file contains the implementation of the class. Each pair of files contains only one class. Any subclass used by a primary class is defined in separate files.

Each one of the ground station's programs was created using Visual C++'s ClassWizard. This merely simplifies the interface into each one of the programs functions or class methods. Most of the code was hand generated, although some of it was machine generated by the ClassWizard. After loading the program source into the Visual C++ editor, invoking up the ClassWizard will display all the class methods that were implemented. Further, all the possible methods under the MFC architecture are additionally listed. By choosing an implemented method, the source for that method is displayed. If a new method is created, the ClassWizard will automatically create the blank

function with all the required overhead for that particular instance. The area for the developer to add the new relevant code is clearly marked. Essentially, for learning the structure of a program, modifying existing code, or adding new methods to the classes in a program, the ClassWizard is an invaluable tool.

One further note on the ground station software. Due to its networking nature, the ground station software must be executed on an Windows NT platform. Windows 95 does not support the required networking elements used by the programs. The result is that the ground station will crash on Windows 95.

The software for the satellite is incorporated into a single .EXE file, called "US.EXE." Furthermore, the source code is not broken into classes since it is written entirely in C. A pseudo class organization is used, however. The source for each major function point of the program is grouped into a separate file, which is labeled according to the functionality contained therein.

Unfortunately, since the MS-DOS like operating system is used, only eight character file names can be used. Thus the orbital calculation module is called "orb_cal.c." Once again, the comment heading in each file thoroughly explains the purpose of the module and its integration into the whole program.

The center module for the program is the file "main.c." The function in this module performs as an automata state machine processor. It continually loops, checking the states of its communications connections and internal processing states. Based on these states, and in conjunction with any input received by the system, the appropriate submodule is called. All the other files comprising the satellite User Services are submodules to the "main" function.

LIST OF REFERENCES

Antonio, Franklin, *Keplerian Elements Tutorial, AMSAT, 1997.*

Awad, Maher, *Object-Oriented Technology for Real-Time Systems*, Prentice Hall, Inc., 1996.

Battin, Richard H., *An Introduction to the Mathematics and Methods of Astrodynamics*, American Institude of Aeronautics and Astronautics, 1987.

Bible, Steven R., and Daniel Sakoda, "Petite Amateur Navy Satellite," NPS.

Davidoff, Martin, *The Satellite Experimenter's Handbook*, The American Radio Relay League, 1990.

Horzepa, Steve, *Practical Packet Radio*, First Edition, The American Radio Relay League, 1995.

Kelso, T.S, ed., *Space Track Report #3: Models for Propagation of NORAD Element Sets*, NORAD, 1980.

Kopetz, H., *Software Reliability*, Springer-Verlag, 1979.

Kreutzfeld, Robert J., and Neese, Richard E., *Methodology for Cost-Effective Software Fault Tolerance for Mission-Critical Systems*, IEEE AES Systems Magazine, September 1997.

Lawrence, Gregory Wade, *Preliminary PANSAT Ground Station Software Design and Use of an Expert System to Analyze Telemetry*, NPS thesis, March 1994.

Lee, P.A., and Anderson, T., *Fault Tolerance Principles and Practice*, Springer-Verlag, 1990.

McGhee, Robert, CS4314 (Symbolic Computing) Class Notes, Winter 1998.

Nelson, Mark and Gailly, Jean-Loup, *The Data Compression Book*, M&T Books, 1996.

Pradhan, D.K., ed., *Fault-Tolerant Computer: Theory and Techniques*, Volume II, Prentice-Hall, 1986.

Severson, Fred J., *An Overview of the Petite Amateur Navy Satellite (PANSAT) Project, NPS* thesis, December 1995.

Shing, Man-Tak, CS3460 (Software Methodology) Class Notes, Fall 1996.

Shing, Man-Tak, CS4580 (Design of Embedded Real-Time Systems) Class Notes, Winter 1997.

Ward, Jeff and Price, Harold E., *PACSAT Protocol Suite: An Overview*, Space Systems Academic Group Dissertation Holdings.

INITIAL DISTRIBUTION LIST

		Number of Copies
1.	Defense Technical Information Center 8725 John J. Kingman Road, Ste 0922 Ft. Belvoir, VA 22060-6218	2
2.	Dudley Knox Library Naval Postgraduate School Monterey, CA 93943-5002	2
3.	ECJ6-NP HQ USEUCOM Unit 30400 Box 1000 APO AE 09128	1
4.	Chairman, Code CS Computer Science Department Naval Postgraduate School Monterey, CA 93943-5101	1
5.	Man-Tak Shing, Code CS/SH Department of Computer Science Naval Postgraduate School Monterey, CA 93943-5101	1
6.	Chairman, Code SP Space Systems Academic Group Naval Postgraduate School Monterey, CA 93943-5002	1
7.	Daniel Sakoda, Code SP/DS Space Systems Academic Group Naval Postgraduate School Monterey, CA 93943-5002 1	1
8.	James Horning, Code SP/JH Space Systems Academic Group Naval Postgraduate School Monterey, CA 93943-5002	1

9. LT Ken Hunter 1
940 Gravenstein Hwy S
Sebastopol, CA 95472

10. MAJ Nelson Ludlow 1
5928 South 4075 West
Roy, UT 84067

11. CDR Gus Lott, Code EC/LT 1
Naval Postgraduate School
Monterey, CA 93943-5121

12. Commanding Officer, Code 30 2
Naval Information Warfare Activity
9800 Savage Rd.
Ft. Meade, MD 20755-6000

DUDLEY KNOX LIBRARY
NAVAL POSTGRADUATE SCH
MONTEREY CA 93943-5101

15 483NPG TH 3369
10/99 22527-200

DUDLEY KNOX LIBRARY
NAVAL POSTGRADUATE SCHOOL
MONTEREY CA 93943-5101

DUDLEY KNOX LIBRARY
3 2768 00357398 1